Peripheral Files Programming Commands

Peripheral Files Programming Commands

Version March, 11 2011

This document describes the commands which are used to write peripheral files. This allows to display/ manipulate configuration registers and the on-chip peripheral registers at a logical level. Registers and their contents are visible and accessible in **PER** window.

Overview

Peripherals in MCU can be displayed and manipulated with the **PER** commands. TRACE32 offers configurable window for displaying memory or I/O structures. Displaying the state of peripheral components or memory based structures is very comfortable.

User can define 'chip macros' and put them together to generate 'project files'. These files describe the port structure for a specific hardware system.

Examples for different microcontrollers reside in the directory **../demo/per**.

Peripheral File General Description

To start writing the peripheral file, please create a file with extension *.per.
".per" is the TRACE32 standard extension for peripheral files.

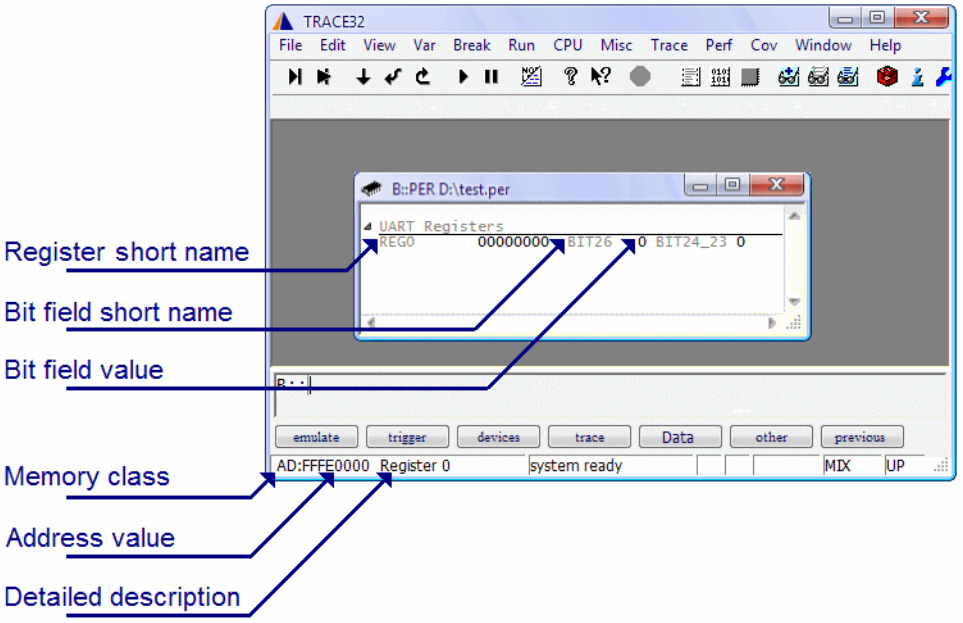
The syntax of a peripheral file is line oriented. Blanks and empty lines can be inserted to define the structure of the program. Comment lines start with semicolon.

Examples of the peripheral file reside in the directory **../demo/per**.

At the beginning of the file, the commands **WIDTH** and **CONFIG** should be placed. The next step is to define the base address using **BASE** command. Each implemented module has to be started with **TREE** command and ended with the **TREE.END** command.

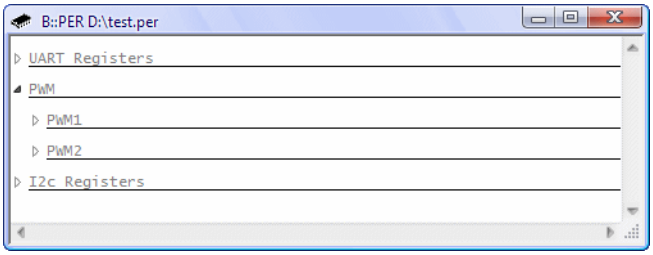
A typical peripheral file implementation is showed below:

```
CONFIG 16. 8.                              ; "dots" mean decimal format
WIDTH 0xb                                  ; 0x means hex format

TREE "Module Registers"                    ; "Treeview" of the module

BASE ad:0xf0000000                         ; base address of the module

GROUP.LONG 0x00++0x3                       ; GROUP definition
  LINE.LONG 0x00 "REG0,Register 0"         ; register definition

    ; one bit filed definition
    BITFLD.LONG 0x00 26. " BIT26 ,Bit 26" "0,1"

    ; 2-bit field definition
    BITFLD.LONG 0x00 23.--24. " BIT24_23 ,Bits 24 to 23" "0,1,2,3"

TREE.END                                   ; end of the tree
```

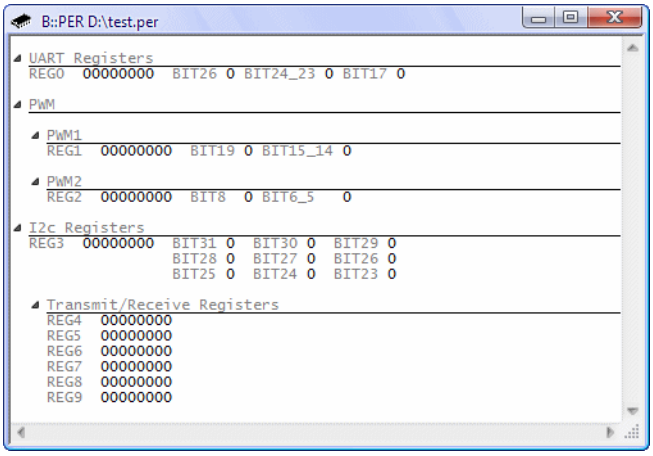

TRACE32
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
B::PER D:\test.per
UART Registers
REG0 00000000 BIT26 0 BIT24_23 0
B::
emulate trigger devices trace Data other previous
AD:FFFE0000 Register 0
system ready
MIX
UP
Register short name
Bit field short name
Bit field value
Memory class
Address value
Detailed description

```
TREE "UART Registers"
    BASE ad:0xfffe0000
    GROUP.LONG 0x00++0x3
         LINE.LONG 0x00 "REG0,Register 0"
              BITFLD.LONG 0x00 26. " BIT26 ,Bit 26" "0,1"
              BITFLD.LONG 0x00 23.--24. " BIT24_23 ,Bits 24 to 23" "0,1,2,3"
              BITFLD.LONG 0x00 26. " BIT17 ,Bit 17" "0,1"
TREE.END

TREE.OPEN "PWM"
    TREE "PWM1"
         BASE ad:0xfffe1000
         GROUP.LONG 0x00++0x3
              LINE.LONG 0x00 "REG1,Register 1"
                   BITFLD.LONG 0x00 19. " BIT19 ,Bit 19" "0,1"
                   BITFLD.LONG 0x00 14.--15. " BIT15_14 ,Bits 15 to 14" "0,1,2,3"
    TREE.END
    TREE "PWM2"
         BASE ad:0xfffe2000
         GROUP.LONG 0x00++0x3
              LINE.LONG 0x00 "REG2,Register 2"
                   BITFLD.LONG 0x00 8. " BIT8  ,Bit 8" "0,1"
                   BITFLD.LONG 0x00 5.--6. " BIT6_5   ,Bits 6 to 5" "0,1,2,3"
    TREE.END
TREE.END
TREE "I2c Registers"
    BASE ad:0xfffe3000
    GROUP.LONG 0x00++0x3
         LINE.LONG 0x00 "REG3,Register 3"
              BITFLD.LONG 0x00 31. " BIT31 ,Bit 31" "0,1"
              BITFLD.LONG 0x00 30. "  BIT30 ,Bit 30" "0,1"
              BITFLD.LONG 0x00 29. "  BIT29 ,Bit 29" "0,1"
              TEXTLINE "                 "
              BITFLD.LONG 0x00 28. " BIT28 ,Bit 28" "0,1"
              BITFLD.LONG 0x00 27. "  BIT27 ,Bit 27" "0,1"
              BITFLD.LONG 0x00 26. "  BIT26 ,Bit 26" "0,1"
              TEXTLINE "                 "
              BITFLD.LONG 0x00 25. " BIT25 ,Bit 25" "0,1"
              BITFLD.LONG 0x00 24. "  BIT24 ,Bit 24" "0,1"
              BITFLD.LONG 0x00 23. "  BIT23 ,Bit 23" "0,1"
    TREE "Transmit/Receive Registers"
         GROUP.LONG 0x10++0x17
              LINE.LONG 0x00 "REG4,Register 4"
              LINE.LONG 0x04 "REG5,Register 5"
              LINE.LONG 0x08 "REG6,Register 6"
              LINE.LONG 0x0c "REG7,Register 7"
              LINE.LONG 0x10 "REG8,Register 8"
              LINE.LONG 0x14 "REG9,Register 9"
    TREE.END
TREE.END
```

Peripheral modules are organized in a tree structure.



Contents of peripheral modules is also organized in a tree structure.



Memory Classes

Format:	<*class*>:<*base_address*>
<*class*>:	Appropriate access method to memory class (**D**, **SD**, **A**, **AD**, **AP**, **ANC**,**DC**, **IC**, **NC**, **ED**, **EAD**, **VM**, **P, etc.**)
<*base_address*>:	Base address of the peripheral module.

Refer to the **"General Commands Reference Guide D" - Memory Classes.**

GROUP Commands

The GROUP commands describe how data is basically read or wirtten to/from memory.

GROUP — Define read/write GROUP

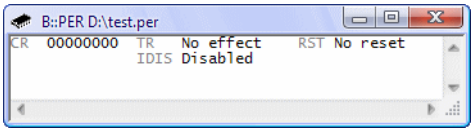
Format:	**GROUP.**<*size*> <*datagrp*>|<*fifogroup*> [**"**<*name*>**"**]
<*size*>:	Size of registers (quad, long, tbyte, word, byte).
<*datagrp*>:	<*address*>++<*number_of_read_bytes-1*> or <*start_address*>--<*end_address*>
<*fifogroup*>:	<*address*> <*address_range*>
<*name*>:	Optional text.

The GROUP commands control the debugger access to the target memory.

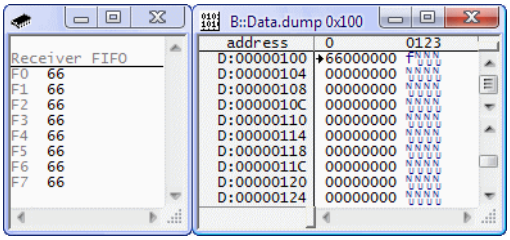
If a name is given, the GROUP is separated from the previous lines and the name is used as headline in the per window. Using numerical values (without memory access class) in adress paramter, the address is calculated by the entered value plus the base address (defined by the last **BASE** command). The GROUP can either use normal memory access or fifo access (reads all bytes from the same address). The whole address range of the GROUP command is read at once. Reading from reserved address range may cause a bus error.

```
BASE ud:0x200                    ;databytes at address sd:0x100--0x101
GROUP sd:0x100--0x101 "PortA"


GROUP 0x50--0x51                 ;databytes at address ud:0x250--0x251


GROUP.LONG sd:0x60--0x6f         ;read memory with 32-bit access


GROUP sd:0x300 0x10              ;fifo at location sd:0x300, 16 bytes
                                 deep
GROUP 0x10 0x4                   ;fifo at ud:0x210, 4 bytes deep
```

```
BASE ad:0x00000000
GROUP 0x00++0x03
   LINE.LONG 0x00 "CR,Control Register"
        BITFLD.LONG 0x00 24. " TR   ,Transfer" "No effect,Transferred"
        BITFLD.LONG 0x00 5. "  RST ,Software Reset" "No reset,Reset"
        TEXTLINE "              "
        BITFLD.LONG 0x00 1. " IDIS ,Interrupt Enable" "Disabled,Enabled"
```




```
BASE ad:0x00000000
GROUP.BYTE 0x100 0x8 "Receiver FIFO"
   LINE.BYTE 0x0 "F0,FIFO position 0"
   LINE.BYTE 0x1 "F1,FIFO position 1"
   LINE.BYTE 0x2 "F2,FIFO position 2"
   LINE.BYTE 0x3 "F3,FIFO position 3"
   LINE.BYTE 0x4 "F4,FIFO position 4"
   LINE.BYTE 0x5 "F5,FIFO position 5"
   LINE.BYTE 0x6 "F6,FIFO position 6"
   LINE.BYTE 0x7 "F7,FIFO position 7"
```



See also

OUT
SAVEOUT

HGROUP — Define read-once/write GROUP

Format:	**HGROUP.**<*size*> <*datagrp*>|<*fifogroup*>[**"**<*name*>**"**]
<*size*>:	Size of registers (quad, long, tbyte, word, byte).
<*datagrp*>:	<*address*>++<*number_of_read_bytes-1*> or <*start_address*>--<*end_address*>
<*fifogroup*>:	<*address*> <*address_range*>
<*name*>:	Optional text.

Similar to GROUP, but this definition is useful for ports which are cleared by a read access. Refer to the **GROUP** command description. HGROUP command prevents target memory from the periodic read access and is useful for 'write-only' ports. In hidden GROUPs only hidden elements e.g. **HIDE** command should be used.

RGROUP — Define read-only GROUP

Format:	**RGROUP.**<*size*> <*datagrp*>|<*fifogroup*> [**"**<*name*>**"**]
<*size*>:	Size of registers (quad, long, tbyte, word, byte).
<*datagrp*>:	<*address*>++<*number_of_read_bytes-1*> or <*start_address*>--<*end_address*>
<*fifogroup*>:	<*address*> <*address_range*>
<*name*>:	Optional text.

Similar to GROUP, but this definition is useful for 'read-only' ports. Refer to the **GROUP** command description.

See also

OUT
SAVEOUT

Format:	**SGROUP** [**"**<*name*>**"**]
<*name*>:	Optional text.

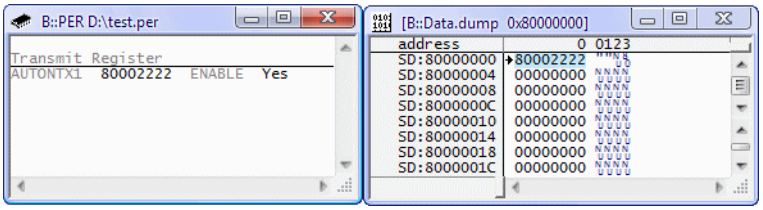
Sequence of memory accesses done to get/set the data.

Usually GROUP commands specify the target memory accesses and the following commands e.g. BITFLD, HEXMASK, etc. define how the data are displayed in the per window.
With SGROUP data is not accessed with **SGROUP** itself, but by a sequence of special commands, which transfer data from memory to the "SGROUP data buffer" or from the "SGROUP data buffer" back to memory. The size of the buffer is 256 bytes.
Afterwards this sequence of special commands the data in the buffer can be displayed by following commands e.g. BITFLD, HEXMASK.

To read/write data from/to memory to/from SGROUP buffer you can use the following commands (which are only allowed in **SGROUP**s):

Command	**Function**
SET <*address*> %<*format*> <*value*>	constant value **-->** memory(address)
SETX <*address*> %<*format*> <*index*>	buffer(index) **-->** memory(address)
GETX <*address*> %<*format*> <*index*>	memory(address) **-->** buffer(index)
CONSTX <*index*> %<*format*> <*value*>	constant value **-->** buffer(index)
VARX <*index*> %<*format*> <*expression*>	variable value **-->** buffer(index)
WRITEBACK	Separate write part from a read part

```
SGROUP "Transmit Register"              ; define sequence GROUP
GETX d:0x80000000 %l 0                  ; read data at 0x80000000 and store
                                        ; them in buffer + offset 0
WRITEBACK                               ; next commands only done for
CONSTX 2 %w 0x2222                      ; per.set
                                        ; write 0x2222 to buffer + offset 2
SETX d:0x80000000 %l 0                  ; write data from buffer + offset 0
                                        ; to memory at 0x80000000
LINE.LONG 0x0                           ; display AUTONTX1 register with
"AUTONTX1,Autonegotiation Next          ; contents of buffer[0…3]
Page Transmit Register 1"
BITFLD.LONG 0 31. "ENABLE" "No,Yes"     ; define bit "Enable"
```



SET — Write constant value to memory

Format:	**SET** <*address*> **%**<*format*> <*value*>
<*address*>:	Target address.
<*format*>:	Defines specific format (quad, long, tbyte, word, byte, le, be).
<*value*>:	Constant value. The value may be a hexadecimal o mask or binary mask. (E.g.: `0yxxxx10xx`)

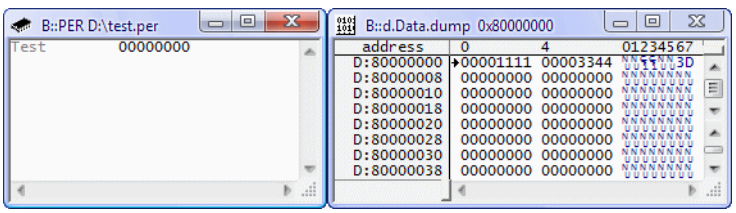
SET command writes data to memory.

The given value is written to the target memory at the specified address or at the base address with added offset. The specified value is written continuously.

Command is only allowed in **SGROUP**.

Example

```
BASE d:0x80000000                       ; set base address to d:0x80000000
SGROUP                                  ; define sequence GROUP
SET d:0x80000000 %l 0x1111              ; write 0x1111 to d:80000000
SET 4 %l 0x3344                         ; write 0x3344 to base address
                                        ; (d:80000000) + offset 4

LINE.LONG 0x0 "Test,Test Register"
```



Format:	**SETX** <*address*> **%**<*format*> <*index*>
<*address*>:	Target address.
<*format*>:	Defines specific format (quad, long, tbyte, word, byte).
<*index*>:	Constant value.

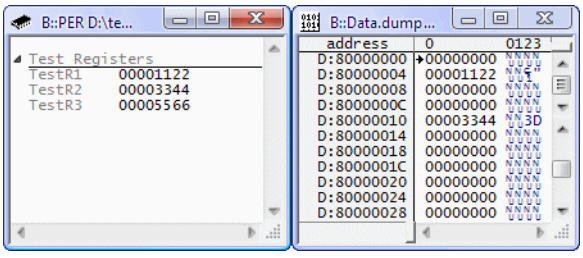
SETX command writes a buffered value to the memory.

A value stored in a buffer at the given buffer offset is written to the target memory at the specified address or base address with added offset.The value is written only once.

Command is only allowed in **SGROUP**.

Example:.

```
CONFIG 16. 8.
WIDTH  10.
BASE 0x80000000
TREE "Test Registers"
;write into buffer : 0x1122 at offet [0], 0x3344 at [4], 0x5566 at [8]
SGROUP

CONSTX 0 %l 0x1122
CONSTX 4 %l 0x3344
CONSTX 8 %l 0x5566
   LINE.LONG 0x0 "TestR1,Test Register 1"
   LINE.LONG 0x4 "TestR2,Test Register 2"
   LINE.LONG 0x8 "TestR3,Test Register 3"

;write buffer contents into target memory : [0..3] at 0x80000004,...
SETX 4    %l 0
SETX 0x10 %l 4
TREE.END
```



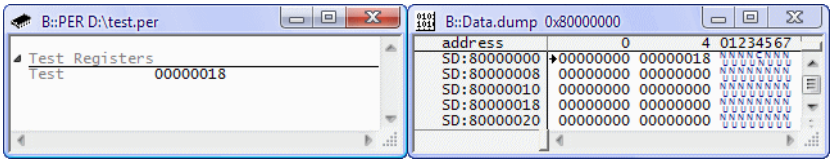
Format:	**GETX** <*address*> **%**<*format*> <*index*>
<*address*>:	Target address equals base address + offset.
<*format*>:	Defines specific format (quad, long, tbyte, word, byte).
<*index*>:	Defines buffer number.

GETX command reads data from the memory and puts it to the buffer. The memory contents from the given address is read using specified access width format. The read data is stored in a buffer at the defined offset.

Command is only allowed in **SGROUP**.

Example:

```
BASE d:0x80000000
TREE "Test Registers"
SGROUP                                  ; define sequence GROUP
SET d:0x80000004 %l 0x18                ; write value 0x18 to target memory
                                        ; at d:80000004
GETX 4 %l 0                             ; read out target memory at base
                                        ; adress d:80000000+offset 4 and
                                        ; store it at buffer+offset 0
LINE.LONG 0x0 "Test,Test Register"      ; display data of buffer[0…3]
TREE.END
```



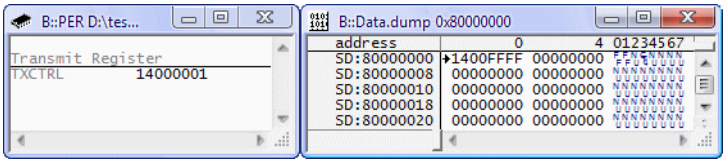
Format:	**CONSTX** <*index*> **%**<*format*> <*value*>
<*index*>:	Defines indexed offset.
<*format*>:	Defines specific format (quad, long, tbyte, word, byte, le, be).
<*value*>:	Defines a constant value. The value may be a hexadecimal or mask or binary mask. (E.g.: `0yxxxx10xx`)

CONSTX command writes a constant value to the buffer. This Data is **not** written to the target memory. The data can be displayed with a following line command.

Command is only allowed in **SGROUP**.

Example:

```
SGROUP "Transmit Register"          ; define sequence GROUP
SET 0x80000000 %l 0x1400ffff        ; write value 1400ffff to target
                                    ; memory at d:80000000

GETX  d:0x80000000 %l 0x00          ; read out target memory at 80000000
                                    ; and store it at buffer + offset 0
CONSTX 2 %w 0x1                     ; write 0x0001 at buffer + offset 2
LINE.LONG 0x0 "TXCTRL,Transmit      ; display data of buffer[0…3]
Control Register"
```



Format:	**VARX** <*index*> **%**<*format*> <*expression*>
<*index*>:	Defines indexed offset.
<*format*>:	Defines specific format (quad, long, tbyte, word, byte, le, be).
<*expression*>:	Defines a PRACTICE expression. The expression will be parsed whenever the PER window updates and its result will be assigned to the SGROUP buffer

VARX command writes a variable value to the SGROUP buffer. This data is **not** written to the target memory. The data can be displayed with a following line command.

The **VARX** command is very similar to the CONSTX command. However the value, which should be assigned to the SGROUP buffer may be based on PRACTICE functions, whose values may change during the display of the PER window.
With **VARX** you can modifiy the SGROUP buffer in any way you like by using the follwing PRACTICE funtions, which access the SGROUP buffer:

PER.BUFFER.BYTE(<*index*>**)** **PER.B.B(**<*index*>**)**	Returns a byte at position <index> from the SGROUP buffer.
PER.BUFFER.WORD(<*index*>**)** **PER.B.W(**<*index*>**)**	Returns a 16 bit word at position <index> from the SGROUP buffer.
PER.BUFFER.LONG(<*index*>**)** **PER.B.L(**<*index*>**)**	Returns a 32 bit word at position <index> from the SGROUP buffer.
PER.BUFFER.QUAD(<*index*>**)** **PER.B.Q(**<*index*>**)**	Returns a 64 bit at position <index> from the SGROUP buffer.

Due to perfomance reasons you should use **VARX** only, if there is no other solution possible.

Command is only allowed in **SGROUP**.

Example:

```
SGROUP "Dummy Counter"                          ; begin Sequence-GROUP
  varx 0 %quad os.timer()                       ; read timer from OS
  varx 9 %q (PER.B.Q(0)/1000.)                  ; define quad data from
                                                  SGROUP buffer at index 0 by
                                                  1000 and store the result
                                                  at index 9 as quad
  textline ""                                   ; diplay data at index 0 as
  decmask.quad 0 0--63. 1 "  milliseconds:"      decimal
  textline ""                                   ; diplay data at index 9 as
  decmask.quad 9 0--63. 1 "  seconds:     "      decimal
  textline ""                                   ; Newline
```

WRITEBACK

Separate write a part from a read part

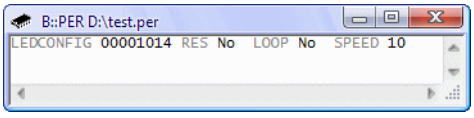
Format:	**WRITEBACK**

Separates the write part of a sequence from the read part.

Command is only allowed in **SGROUP**.

:

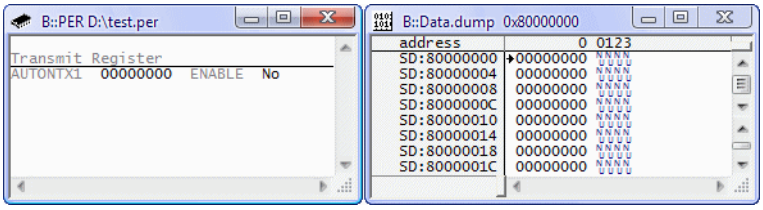
```
SGROUP
   SET 0 %l 0x1014
   GETX 0 %l 0
   WRITEBACK
   CONSTX 2 %w 0x2014
   SETX 0 %l 0
   LINE.LONG 0x0 "LEDCONFIG,LED Configuration Register (20)"
        BITFLD.LONG 0x0 31. "RES ,Reset" "No,Yes"
        BITFLD.LONG 0x0 30. " LOOP ,Loopback" "No,Yes"
        BITFLD.LONG 0x0 29. " SPEED ,Speed" "10,100"
```



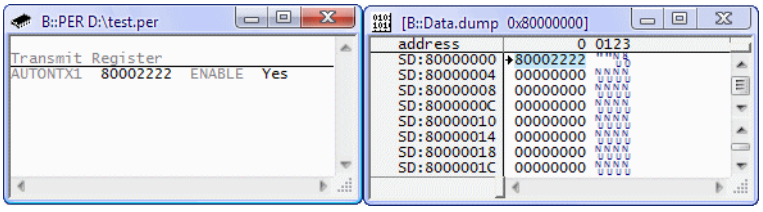
The commands after write back are executed only if **PER.SET** command is used. For displaying the data in the PER-window these commands are ignored.

```
SGROUP "Transmit Register"              ; define sequence GROUP
GETX d:0x80000000 %l 0                  ; read data at 0x80000000 and store
                                        ; them in buffer + offset 0

WRITEBACK                               ; next commands only executed, if a
                                        ; write access is done in per-window

CONSTX 2 %w 0x2222                      ; write 0x2222 to buffer + offset 2
SETX d:0x80000000 %l 0                  ; write data from buffer + offset 0
                                        ; to memory at 0x80000000

LINE.LONG 0x0 "AUTONTX1,Transmit        ; display AUTONTX1 register with
Reg."                                   ; contents of buffer[0…3]
BITFLD.LONG 0 31. "ENABLE "             ; if bit 31 is changed/written
"No,Yes"                                ; constx and setx are done
```

Opening the per-window results in displaying data from memory.



Changing state of the ENABLE bit results also in writing constant value 0x2222 to the register.



Format:	**WSGROUP.**<*size*> <*wr_acc_addr*> <*rd_acc_addr*>
<*size*>:	Size of registers (quad, long, tbyte, word, byte).
<*wr_acc_addr*>:	Address of the register where data is to be written into.
<*rd_acc_addr*>:	Address of the register where data is to be read from.

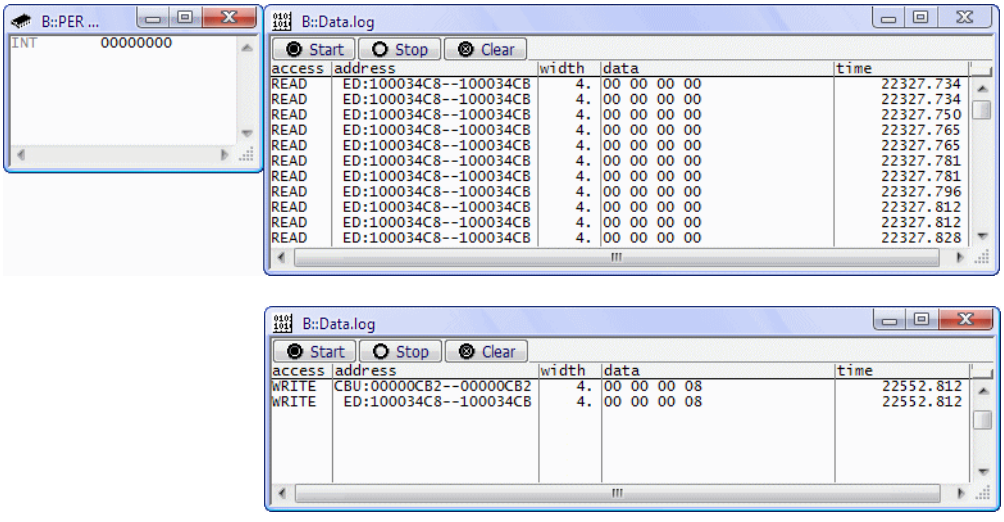
WSGROUP is a specific GROUP command, which forces the debugger to access different registers for read and for write accesses. It is only useful, if the core has write-only registers and their contents are duplicated in shadow registers, which are read- and writable.

Read-/write accesses have following effects:

- write access: Data is written to write-only registers (dataGROUP) as well as to the shadow registers.

- read access: Data is read from the shadow registers.

```
WSGROUP.LONG (ecbu:0x0CB2)++0 (ed:0x100034C8)
   LINE.LONG 0x0 "INT,Self-interrupt register"
```



Format:	**WGROUP.**<*size*> <*datagrp*>|<*fifogroup*>[**"**<*name*>**"**]
<*size*>:	Size of registers (quad, long, tbyte, word, byte).
<*datagrp*>:	<*address*>++<*number_of_read_bytes*-1> or <*start_address*>--<*end_address*>
<*fifogroup*>:	<*address*> <*address_range*>
<*name*>:	Optional text.

Similar to GROUP command. This definition is useful for 'write-only' ports. The current state of the port is held in the emulation memory (must be mapped at this location). Refer to the **GROUP** command description.

```
WGROUP sd:0x50--0x51          ;the port at address sd:0x50--0x51
                              ;is a write-only port (e.g. 74xx374)
                              ;but the state can be read via
                              ;dual-port access
```

ASSERT — Aboard if condition not met

Format:	**ASSERT** <*expression*> [<*string*>]
<*expression*>:	Expression which must evaluate to a boolean. If the result of the expression is FALSE, the parsing of the PER file will be stopped and an error message will be shown.
<*string*>:	Optional string containing an error message, which will be shown if <*expression*> evaluates to FALSE.

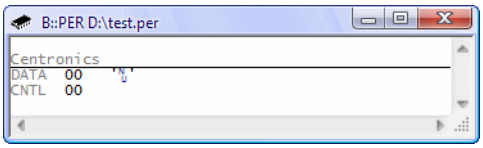
With **ASSERT** you can ensure that your environment meets a certain condition, before TRACE32 should go on with the parsing of the PER file.

If you omit the optional string with an error message, the following message will be shown instead:
`Assertion failed: <expression>`

Example: Ensure that a PER file is only parsed by “TRACE32 for ARM”

```
assert cpufamily()=="ARM" "Sorry, this PER file is only for ARM cores"
```

Format:	**BASE** <*address*>
<*address*>:	Start address of the peripheral module.



This command sets the start address for the peripheral module and refers to simple offset ranges. This expression is permanently recalculated. If the parameters contain functions or symbols, it reflects later changes in the parameters. The BASE command specifies memory class which is responsible for setting appropriate addressing mode. Memory classes are described in **Memory Classes** section.

```
;use fixed base
BASE d:0xffff0000
   GROUP.LONG 0x00++0x3
        LINE.LONG 0x00 "Reg_0,Register 0"

; use variable base
BASE (SYS.BASE()&0x0f)*0x1000

; use variable base
BASE Data.Long(base_pointer)
```

Format:	**BASEOUT** <*data_address*> <*address*> <*dataread*> <*datawrite*>
<*data_address*>:	Data register address.
<*address*>:	Indirect base address.
<*dataread*>:	Data read from the specified port.
<*datawrite*>:	Data write to the specified port.

Sends specified data to the port. BASEOUT is a combination of BASE and OUT commands and must be placed before a GROUP definition. The data is sent to the indirect addressed port prior to the port access or modification. If two bytes are defined, the second byte is used for writing to the specified port (different indices for reading and writing). It is useful for ports which must be selected first.

Please consider: As the display is refreshed permanently the index register is modified as well. In most cases this error occurs.

```
BASEOUT a:0x104 a:0x100 0x01                    ; indirect base and
GROUP.LONG 0x00++0x03                           ; data register address
   LINE.LONG 0x0 "REG1,Register index 1"
```

See also

BASESAVEOUT
OUT
SAVEOUT

Format:	**BASESAVEOUT** <*data_address*> <*address*> %<format> <*dataread*> <*datawrite*>
<*data_address*>:	Data register address.
<*address*>:	Indirect base address.
<format>:	Defines specific format (quad, long, tbyte, word, byte).
<*dataread*>:	Data read from the specified port.
<*datawrite*>:	Data write to the specified port.

Sends specified data to the port. BASESAVEOUT is a combination of BASE and SAVEOUT commands and must be placed before a GROUP definition. The data is sent to the indirect addressed port prior to the port access or modification. The current values at the port are read before the access is made and are restored after the access. If two bytes are defined, the second byte will be used for writing to the specified port (different indices for reading and writing). This is useful for ports which are selected by another port when the index register can be read back.

```
BASESAVEOUT a:0x104 a:0x100 %long 0x01 0x01    ; indirect base and
GROUP.LONG 0x00++0x03                          ; data register address
   LINE.LONG 0x0 "REG1,Register index 1"
```

See also

BASEOUT
OUT
SAVEOUT

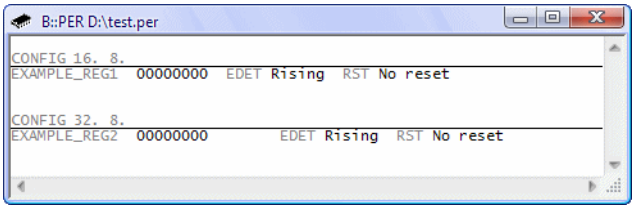
Format:	**CONFIG** <*width*> [<*bitsperline*>]
<*width*>:	Width of displayed bits.
<*bitsperline*>:	Number bits per line.

As a default the display of the bits is configured to be 8 bits wide. With this command the width can be changed to other values. The width can be changed frequently and anywhere in the definition.

```
CONFIG 16. 8.
```

Allows bit by bit display of 16-bit ports, with eight bits per line.

```
CONFIG 16. 8.
WIDTH 0xe

BASE d:0x00000000
GROUP.LONG 0x00++0x03
LINE.LONG 0x00 "EXAMPLE_REG1,Example Register 1"
BITFLD.LONG 0x00 31. " EDET ,Edge detection" "Rising,Falling"
BITFLD.LONG 0x00 24. " RST ,Transfer" "No reset,Reset"

CONFIG 32. 8.
GROUP.LONG 0x04++0x03
LINE.LONG 0x00 "EXAMPLE_REG2,Example Register 2"
BITFLD.LONG 0x00 31. " EDET ,Edge detection" "Rising,Falling"
BITFLD.LONG 0x00 24. " RST ,Transfer" "No reset,Reset"
```



ELSE — Conditional GROUP display

Refer to the **IF** command.

ELIF — Conditional GROUP display

Refer to the **IF** command.

ENDIAN — Define little or big endian

Format:	**ENDIAN** [BE | LE | DEF]

With DEF parameter the endianess is set due to the configuration of the debugger. With this command the debugger accesses the target data with the specified endianess. This is done independent of the target and the system endianess settings.

Default: **ENDIAN** DEF

```
ENDIAN.LE                              ; little endian
ENDIAN.BE                              ; big endian
ENDIAN.DEF                             ; target default endian
```

ENDIF — Conditional GROUP display

Refer to the **IF** command

Format:	**HELP.Winhelp "**<*filename*>,<*item*>**"** **HELP.Online "**<*item*>**"**

Defines a button in the last GROUP header or tree control. HELP.Online calls the TRACE32 online manual. HELP.Winhelp calls a windows help file (available on Windows only).

IF — Conditional GROUP display

Format:	**IF** <*condition*>
	ELIF <*condition*>
	ELSE
	ENDIF

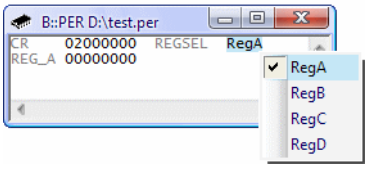
<*condition*>:	Condition examples: - eval()==<*condition_val*> - %<*parameter*>==<*condition_val*> - (((data.<*size*>(<*address*>))&<*bit_mask*>)==<*condition_val*>)

GROUPs can be displayed conditionally using IF...ENDIF commands. GROUPs defined in different IF and ELIF statements are overlaid at the same place in the window. Only GROUPs which reside within the fulfilled condition are displayed. The ELSE part is displayed only when no other condition is true. All conditions are dynamically recalculated to reflect the current state of the peripheral.

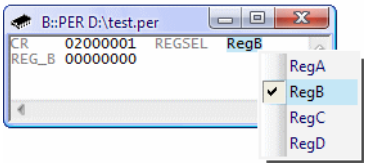
Note: The **IF** command cannot be used inside a **GROUP**. (Please use **IF** always before a new GROUP.)

```
IF (((Data.Long(d:0x00))&0xf)==0x0)
  GROUP.LONG d:0x0++0x7
    LINE.LONG 0x0 "CR,Control register"
      BITFLD.LONG 0x0 0.--1. " REGSEL  ,Register select" "RegA,RegB,RegC,RegD"
    LINE.LONG 0x4 "REG_A,Register A"
ELIF (((Data.Long(d:0x00))&0xf)==0x1)
  GROUP.LONG d:0x0++0x7
    LINE.LONG 0x0 "CR,Control register"
      BITFLD.LONG 0x0 0.--1. " REGSEL  ,Register select" "RegA,RegB,RegC,RegD"
    LINE.LONG 0x4 "REG_B,Register B"
ELIF (((Data.Long(d:0x00))&0xf)==0x2)
  GROUP.LONG d:0x0++0x7
    LINE.LONG 0x0 "CR,Control register"
      BITFLD.LONG 0x0 0.--1. " REGSEL  ,Register select" "RegA,RegB,RegC,RegD"
    LINE.LONG 0x4 "REG_C,Register C"
ELSE
  GROUP.LONG d:0x0++0x7
    LINE.LONG 0x0 "CR,Control register"
      BITFLD.LONG 0x0 0.--1. " REGSEL  ,Register select" "RegA,RegB,RegC,RegD"
    LINE.LONG 0x4 "REG_D,Register D"
ENDIF
```

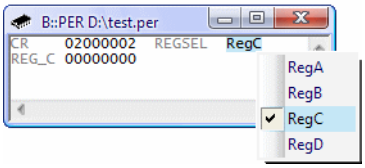
Register REG_A is selected if the value of the REGSEL bit field equals 0.



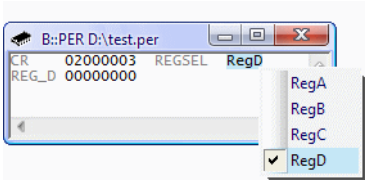
Register REG_B is selected if the value of the REGSEL bit field equals 1.



Register REG_C is selected if the value of the REGSEL bit field equals 2.



Register REG_D is selected if the value of the REGSEL bit field equals 3.



See also

ELSE
ENDIF
SIF

WIDTH — Define field width

Format	**WIDTH** <*register_width*> [<*bit_width*>]
<*register_width*>:	Defines the field width of the register name field, default is 6.
<*bit_width*>:	Optional. Defines the width of one bit field within the [bit array display][link]. Default is 9.

Defines the width of the register name field and the width of one bit in the displayed bit array.

Default: **WIDTH** 6 9

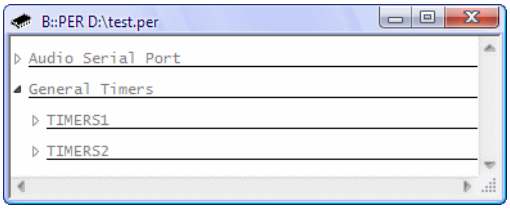
```
WIDTH 16. 8.
```

TREE — Define hierarchic display

Format:	**TREE "**<*name*>**"** **TREE.OPEN "**<*name*>**"** **TREE.END**

Defines a “Treeview” of peripheral modules. The tree can be displayed/hidden by a tree control (+/-). It is possible to nest trees.

```
TREE "Audio Serial Port"          ; tree GROUP displayed closed by default
;                                 ; definition of the GROUP members
TREE.END
TREE.OPEN "General Timers"        ; tree GROUP displayed opened in the
   TREE "TIMERS1"                 ; peripheral window
   ;
   TREE.END
TREE.END
```



SIF — Conditional interpretation

Format:	**SIF** (cpu()==“<*cpu_name*>”) **SIF** (cpuis("<*cpu_name*>*")) **SIF** (<logical_comparison>)

According to the condition a block between **SIF** and **ENDIF** (or **SIF** and **ELSE**) will be interpretated, when the peripheral file is opened or reparsed.

The **SIF** command can be used also inside the GROUPs.

See also

ELSE
ENDIF
IF

Commands within GROUPs

These commands are only usefull inside a GROUP (**GROUP**, **RGROUP**, **WGROUP**, **HGROUP**, **SGROUP**).

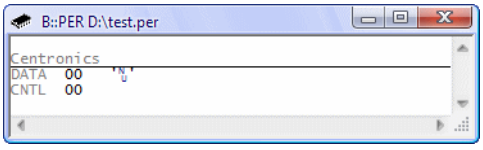
Beside the commands **OUT**, **SAVEOUT** and **BUTTON**, which extend the memory access by a GROUP, the commands define how the data fetched by a GROUP command should be displayed and/or modified.

ASCII — Display ASCII character

Format:	**ASCII**

The previously defined byte is displayed as an **ASCII** character.

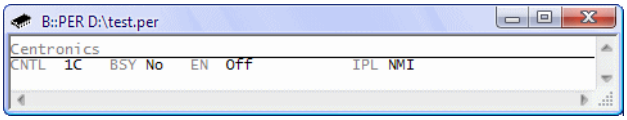
```
GROUP.BYTE sd:0x100--0x101 "Centronics"
   LINE.BYTE 0x0 "DATA,Centronics Data Register"
        ASCII
   LINE.BYTE 0x1 "CNTL,Centronics Control Register"
```



Format:	**BIT** <*bit*>|<*bitrange*> **"**<*short_name*>,<*long_name*>**" "**<*choices*>**"**
<*bit*> | <*bitrange*>:	Defines bit's number and range. LSB is defined as the first, MSB as the second character.
<*short_name*>:	Short name (abbreviation) of corresponding bit.
<*long_name*>:	The sentence accurately describing a bits functionality.
<*choices*>:	Indicates states with bit field may take. LSB is defined as the first, MSB as the last one. Each state is separated by a comma.

These fields are in fixed positions in the per window. The bit numbers must be entered from MSB to LSB. The size of a field depends on the number of bits and the size of the name header.

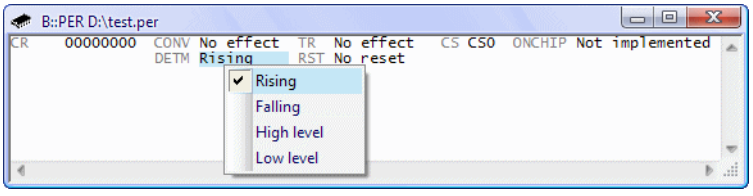
```
GROUP sd:0x100--0x101 "Centronics"
   LINE.BYTE 0x00 "CNTL,Centronics Control Register"
        BIT 7 "BSY,Centronics Busy" "No,Yes"
        BIT 6 "EN,Centronics Enable" "Off,On"
        BIT 2--4 "IPL,Centronics Interrupt Level" "Off,1,2,3,4,5,6,NMI"
```



Format:	**BITFLD.**<*size*> <*offset*> <*bit_range*> "<*short_name*>,<*long_name*>" "<*choices*>"
<*size*>:	Size of register (quad, long, tbyte, word, byte).
<*offset*>:	The bit field offset refers to the start address of the GROUP command.
<*bit_range*>:	Defines range of the bit field. LSB is defined as the first, MSB as the second character. Optionally the third character is bit (or bit range), used if two bit fields are conjuncted.
<*short_name*>:	Short name (abbreviation) of corresponding bit field.
<*long_name*>:	The sentence accurately describing a bit field functionality.
<*choices*>:	Defines the possibles values (in words) which the bit field may take. LSB is defined as the first, MSB as the last one. Each state is separated by a comma.

BITFLD is used to display the bit field name and its contents in a free format. The fields are chained together in a line. A new line can be created by a **TEXTLINE** command.

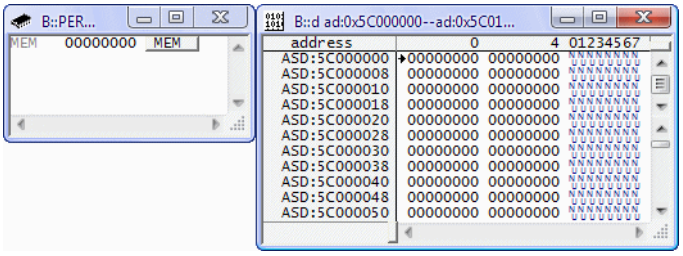
```
BASE d:0x00000000
GROUP 0x00++0x03
      LINE.LONG 0x00 "CR,Control Register"
           BITFLD.LONG 0x00 31. " CONV ,Conversion Bit" "No effect,Conv"
           BITFLD.LONG 0x00 24. " TR ,Transfer" "No effect,Transferred"
           BITFLD.LONG 0x00 16.--19. " CS ,Chip Select"
"CS0,CS1,CS2,CS3,CS4,CS5,CS6,CS7,CS8,CS9,CS10,CS11,CS12,CS13,CS14,CS15"
           BITFLD.LONG 0x00 5. " ONCHIP ,On chip trace implemented" "Not
implemented,Implemented"
TEXTLINE " "
           BITFLD.LONG 0x00 1. 3. " DETM ,Detection mode"
"Rising,Falling,High level,Low level"
           BITFLD.LONG 0x00 0. " RST ,Reset mode" "No reset,Reset"
```



Format:	**BUTTON "**<*text*>**" "**<*commandline*>**"**
<*text*>:	Name of the button.
<*commandline*>:	Contains command, address area and an access size.

Clicking an input-field (button) executes the defined command line. This field can be used to execute input/output commands or open different views (e.g. memory dumps).

```
GROUP.LONG 0x00++0x3
    LINE.LONG 0x00 "MEM,Memory Array"
    BUTTON "MEM " "d ad:0x5C000000--ad:0x5C01FFFF /LONG"
```

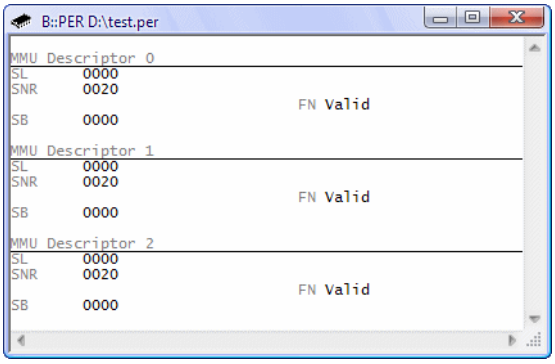


COPY Copy GROUP

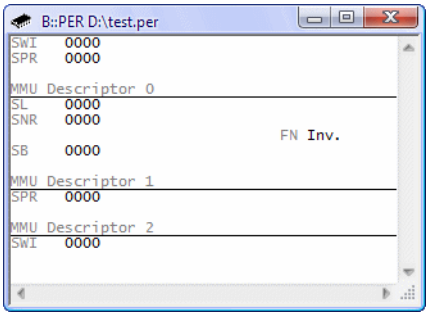
Format:	**COPY** [<*number*>]
<*number*>:	Optional GROUP number.

Copies the last defined GROUP to the current GROUP. The optional argument defines which GROUP should be copied. Number of the GROUP is calculated backward form the current one. The command is used to duplicate the definition of GROUPs,e.g. for devices with many equal channels.

```
GROUP.WORD sd:0x80008038--0x8000803f "MMU Descriptor 0"
    LINE.WORD 0x0 "SL,Segment Length"
    LINE.WORD 0x2 "SNR,Segment Number"
         bit 5 " FN, Flush" "Inv.,Valid"
    LINE.WORD 0x4 "SB,Segment Base Address"
GROUP.WORD sd:0x80008048--0x8000804f "MMU Descriptor 1"
    copy
GROUP.WORD sd:0x80008050--0x80008057 "MMU Descriptor 2"
    COPY
```




```
GROUP.WORD sd:0x80008034--0x80008035
    LINE.WORD 0x0 "SWI,Segment Width"
GROUP.WORD sd:0x80008036--0x80008037
    LINE.WORD 0x0 "SPR,Segment Priority"
GROUP.WORD sd:0x80008038--0x8000803f "MMU Descriptor 0"
    LINE.WORD 0x0 "SL,Segment Length"
    LINE.WORD 0x2 "SNR,Segment Number"
         bit 5 " FN, Flush" "Inv.,Valid"
    LINE.WORD 0x4 "SB,Segment Base Address"
GROUP.WORD sd:0x80008048--0x8000804f "MMU Descriptor 1"
    COPY 2
GROUP.WORD sd:0x80008050--0x80008057 "MMU Descriptor 2"
    COPY 4
```



DECMASK — Define bits for decimal display

Format:	**DECMASK.**<*size*>[.<*length*>] <*offset*> <*bit_range*> <*scale*> [<*add*>] **"**<*short_name*>,<*long_name*>**"**
<*size*>:	Size of register (quad, long, tbyte, word, byte).
<*length*>:	Length of displayed field (quad, long, tbyte, word, byte).
<*offset*>:	The DECMASK field offset refers to the start address of the GROUP command.
<*bit_range*>:	Defines range of the DECMASK field. LSB is defined as the first, MSB as the second character.
<*scale*>:	Multiplier value.
<*add*>:	Optional addend - increases value.
<*short_name*>:	Short name (abbreviation) of corresponding DECMASK field.
<*long_name*>:	The sentence accurately describing a DECMASK field functionality.

While the similar command **HEXMASK** displays bits as a hexadecimal value, **DECMASK** displays bits as deciaml value.

DECMASK defines a set of bits, which should be displayed as decimal value. The bits are extracted from the current buffer at location defined in the bitrange. The result of this extract is multiplied by <scale> and increased by the optional <add> value.

Format:	**EVENTFLD.**<*size*> <*offset*> <*bit_range*> **"**<*short_name*>,<*long_name*>**"** **"**<*choices*>**"**
<*size*>:	Size of register (quad, long, tbyte, word, byte).
<*offset*>:	The event bit offset refers to the start address of the GROUP command.
<*bit_range*>:	Defines range of the bit field. LSB is defined as the first, MSB as the second character. Optionally the third character is bit (or bit range), used if two bit fields are conjuncted.
<*short_name*>:	Short name (abbreviation) of corresponding event bit field.
<*long_name*>:	The sentence accurately describing a event bit field functionality.
<*choices*>:	Indicates states with bit field may take. LSB is defined as the first, MSB as the last one. Each state is separated by a comma.

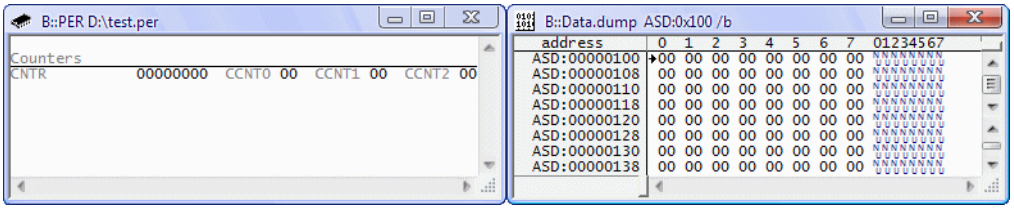
Defines an event bit display in a free format. An event bit can be cleared by writing a '1'. Writing '0' does not affect event bit. The fields are chained together in a line. A new line can be created by a **TEXTLINE** command. The implementation format is the same as a **BITFLD** format.

```
GROUP.WORD d:0x100--0x11f "TPU Channels"
   TEXTLINE ""
   TEXTLINE "CH FUNC PRIO HSF HSR IEF ISF LNK SGL CHS PRM0 PRM1"
   TEXTLINE " 0,Channel 0"
   BITFLD.WORD 0x1e 0.--1. " " " Off, Low, Mid,High"
   BITFLD.WORD 0x16 0.--1. " " " $0, $1, $2, $3"
   EVENTFLD.WORD 0x1a 0. " " "No,Yes"
```

Format:	**HEXFLD.**<*length*> <*offset*> **"**<*short_name*>,<*long_name*>**"**
<*length*>:	Length of HEX field (quad, long, tbyte, word, byte).
<*offset*>:	The HEX field offset refers to the start address of the GROUP command.
<*short_name*>:	Short name (abbreviation) of corresponding HEX field.
<*long_name*>:	The sentence accurately describing a HEX field functionality.

Defines HEX value in a free format. The fields are chained together in a line. A new line can be created using **TEXTLINE** command. If not the whole value should be displayed. The output size can be limited by the “length” parameter.

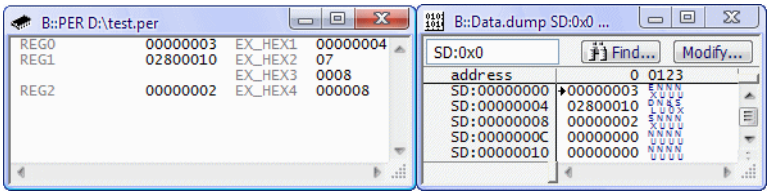
```
GROUP 0x100++0x03 "Counters"
   LINE.LONG 0x00 "CNTR,Channel Counter Register"
        HEXFLD.BYTE 0x00 " CCNT0 ,Channel Counter 0"
        HEXFLD.BYTE 0x01 " CCNT1 ,Channel Counter 1"
        HEXFLD.BYTE 0x02 " CCNT2 ,Channel Counter 2"
```



Format:	**HEXMASK.**<*size*>[.<*length*>] <*offset*> <*bit_range*> <*scale*> [<*add*>] **"**<*short_name*>,<*long_name*>**"**
<*size*>:	Size of register (quad, long, tbyte, word, byte).
<*length*>:	Length of HEX mask field (quad, long, tbyte, word, byte).
<*offset*>:	The HEX mask field offset refers to the start address of the GROUP command.
<*bit_range*>:	Defines range of the HEX mask field. LSB is defined as the first, MSB as the second character.
<*scale*>:	Multiplier value.
<*add*>:	Optional addend - increases Hex mask value.
<*short_name*>:	Short name (abbreviation) of corresponding HEX mask field.
<*long_name*>:	The sentence accurately describing a HEX mask field functionality.

Defines set of bits using HEX value. The bits are extracted from the current buffer at location defined in the bitrange. The result of this extract is multiplied by scale. The <add> value is optional.

```
CONFIG 16. 8.

BASE 0x0
WIDTH 6.
GROUP.LONG 0x00++0xb
LINE.LONG 0x00 " REG0,register 0"
   HEXMASK.LONG 0x00 0.--29. 1. 1. " EX_HEX1  ,Example Hex mask 1"
LINE.LONG 0x04 " REG1,Register 1"
   HEXMASK.LONG.BYTE 0x04 23.--30. 1. 2. " EX_HEX2  ,Example Hex mask 2"
   TEXTLINE "              "
   HEXMASK.LONG.WORD 0x04 4.--15. 8. "  EX_HEX3  ,Example Hex mask 3"
LINE.LONG 0x8 " REG2,Register 2"
   HEXMASK.LONG.TBYTE 0x08 0.--23. 1. 6. " EX_HEX4  ,Example Hex mask 4"
```

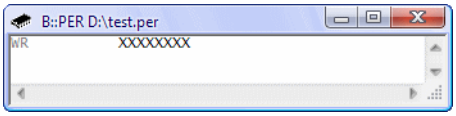


HIDE — Define write-only line

Format:	**HIDE.**<*size*> <*offset*> **"**<*short_name*>,<*long_name*>**"**
<*size*>:	Size of register (quad, long, tbyte, word, byte).
<*offset*>:	The register offset refers to the start address of the hgroup command.
<*short_name*>:	Short name (abbreviation) of corresponding register.
<*long_name*>:	The sentence accurately describing a register functionality.

This field is used for write-only ports like USART transmitters data registers. **HIDE** command should by used together with **HGROUP** command.

```
HGROUP.LONG 0x00++0x3
   HIDE.LONG 0x00 "WR,Write only Register"
```

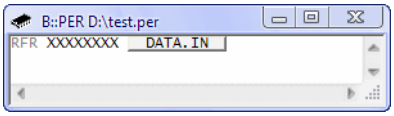


IN — Define input field

Format:	**IN**

An input-field (key) is displayed for the previously defined byte. Clicking that field results in reading data from previously defined location. To execute a read cycle IN command must be used along with a **HIDE** definition. It is used for destructive-read ports (i.e. data port of serial interface).

```
BASE d:0xA00F0000
HGROUP.LONG 0x00++0x3
   HIDE.LONG 0x00 "RFR,Receive FIFO Register"
   IN
```

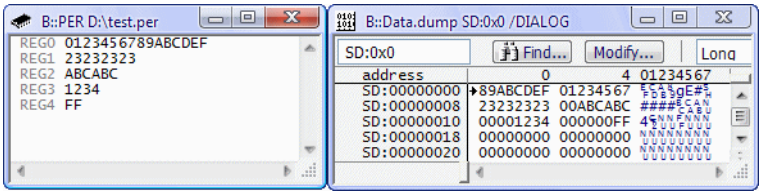


LINE — Define line

Format:	**LINE.**<*size*> <*offset*> **"**<*short_name*>,<*long_name*>**"**
<*size*>:	Size of register (quad, long, tbyte, word, byte).
<*offset*>:	The register offset refers to the start address of the GROUP command.
<*short_name*>:	Short name (abbreviation) of corresponding register.
<*long_name*>:	Register long name (a sentence accurately describing the register functionality)

The LINE command defines registers short name and its long name. The value of the offset is added to the address defined in the previous GROUP command. The **CONFIG** command affects the displayed format of the LINE command

```
BASE 0x0
WIDTH 6.
GROUP.QUAD 0x00++0x7
   LINE.QUAD  0x00 " REG0,Register 0"
GROUP.LONG    0x08++0x3
   LINE.LONG  0x00 " REG1,Register 1"
GROUP.TBYTE   0x0c++0x2
   LINE.TBYTE 0x00 " REG2,Register 2"
GROUP.WORD    0x10++0x1
   LINE.WORD  0x00 " REG3,Register 3"
GROUP.BYTE    0x14++0x0
   LINE.BYTE  0x00 " REG4,Register 4"
```



MUNGING — Translate to little endian mode (PowerPC only)

Format:	**MUNGING** <*be*|*le*>

Usually byte ordering is either little endian or big endian mode. For PPC additional munging little endian and munging big endian modes are provided. For a detailed description refer to PPC documentation.

Special address translation for PowerPC little endian mode.

Note: **MUNGING** is only available on TRACE32 for PowerPC.

```
MUNGING.LE
```

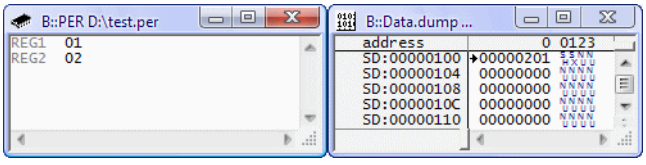
Format:	**OUT** <*address*> <*dataread*> <*datawrite*>
<*address*>:	Destination address.
<*dataread*>:	Data read from the specified port.
<*datawrite*>:	Data write to the specified port.

Sends specified data to the port. OUT command must be placed after a GROUP definition. The data is sent to the port prior to the port access or modification. If two bytes are defined, the second byte is used for writing to the specified port (different indices for reading and writing). It is useful for ports which must be selected first.

Please consider: As the display is refreshed permanently the index register is modified as well. In most cases this error occurs.

Note: **OUT** command has no effect inside an **SGROUP** command.

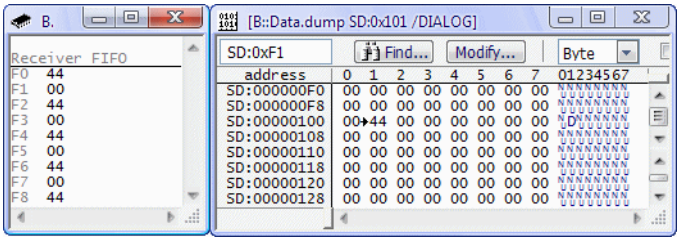
```
GROUP sd:0x100--0x100                          ; select register 1
   OUT sd:0x100 0x01
   LINE.BYTE 0x0 "REG1,Register index 1"
GROUP sd:0x101--0x101                          ; select register 2
   OUT sd:0x101 0x02
   LINE.BYTE 0x0 "REG2,Register index 2"
```



See also

BASEOUT
BASESAVEOUT
SAVEOUT

```
GROUP sd:0x101 0x10 "Receiver FIFO"
OUT sd:0x100 0 0x80 0
      LINE.BYTE 0x0 "F0,FIFO position 0"
      LINE.BYTE 0x1 "F1,FIFO position 1"
      LINE.BYTE 0x2 "F2,FIFO position 2"
      LINE.BYTE 0x3 "F3,FIFO position 3"
      LINE.BYTE 0x4 "F4,FIFO position 4"
      LINE.BYTE 0x5 "F5,FIFO position 5"
      LINE.BYTE 0x6 "F6,FIFO position 6"
      LINE.BYTE 0x7 "F7,FIFO position 7"
      LINE.BYTE 0x8 "F8,FIFO position 8"
```



SAVEOUT — Save original and output a value

Format:	**SAVEOUT** <*address*> **%**<*format*> <*dataread*> <*datawrite*>
<*address*>:	Data address.
<*format*>:	Defines specific format (quad, long, tbyte, word, byte).
<*dataread*>:	Data read from the specified port.
<*datawrite*>:	Data write to the specified port.

Sends the specified data to the port. The current values at the port are read before the access is made and are restored after the access. The byte is sent to the port prior to the port access or modification. SAVEOUT command must be placed after a GROUP definition. If two bytes are defined, the second byte will be used for writing to the specified port (different indices for reading and writing). This is useful for ports which are selected by another port when the index register can be read back.

Note: **OUT** command has no effect inside an **SGROUP** command.

```
GROUP d:0x11--0x11 "SERIAL CONTROL 80196"
   SAVEOUT d:0x14 %byte 0x00 0x0f              ;index 0 for read,
                                               ;15 for write
   LINE.BYTE 0 "SCN,Serial Control Register"
```

See also

BASEOUT
BASESAVEOUT
OUT

SETCLRFLD — Define set/clear locations

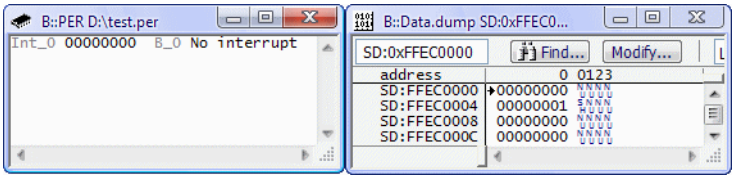
Format:	**SETCLRFLD.**<*size*> <*offset1*> <*bit1*> <*offset2*> <*bit2*> <*offset3*> <*bit3*> **“**<*short_name*>,<*long_name*>**”** **“**<*choices*>**”**
<*size*>:	Size of register (quad, long, tbyte, word, byte).
<*offset1*><*bit1*>:	Status register offset and corresponding bit number.
<*offset2*> <*bit2*>:	Set register offset and corresponding bit number.
<*offset3*> <*bit3*>:	Clear register offset and corresponding bit number.
<*short_name*>:	Short name (abbreviation) of corresponding set/clear bits.
<*long_name*>:	The sentence accurately describing a set/clear bits functionality.
<*choices*>:	Indicates states with bit field may take. The first state is responsible for clearing, the second one for setting corresponding set/clear bits. Each state is separated by a comma.

Defines a bit display in a free format. The fields are chained together in a line. A new line can be created by a **TEXTLINE** command.

The command is an extension of the **BITFLD** command. Additionally to the BITFLD command two further locations must be entered. The first parameter pair offset1 - bit1 is the location where the data is read from. The second parameter pair offset2 - bit2 is the set location. The third parameter pair offset3 - bit3 is the clear location.

Usually the SETCLRFLD-command is used if the read location is a status register, which shows the status of I/O ports and other (not static) registers exist to enable and disable ports. If the port is enabled, the value of '1' is set to the corresponding bit in the register addressed by location 2 (other bits are cleared). If the port is disabled, the value of '1' is set at the corresponding bit position in the register addressed by location 3 (the other bits are cleared).

```
BASE sd:0xffec0000
GROUP.LONG 0x00++0x3
   LINE.LONG 0x00 "Int_0,Interrupt Register 0"
        SETCLRFLD.LONG 0x0 0. 0x4 0. 0x8 0. " B_0 ,Bit 0"
"No Interrupt,Interrupt"

;writing 1 sets the bit in the Set   Register
;writing 0 sets the bit in the Clear Register
;the result is read from the Status register
```



SYSCON — SYSCON register (C166/ST10 only)

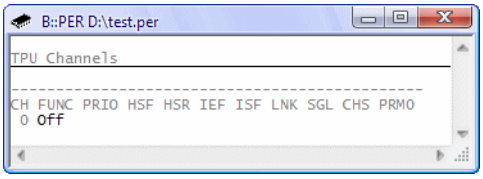
Format:	**SYSCON**

Special block for C166 bondout cpus (ICE).

Format:	**TEXTLINE "***<text>***"**
<text>:	Optional text.

The text can either be used as general comment or as a header to **BITFLD** or **HEXFLD** fields. **TEXTLINE** creates a new line.

```
GROUP d:0x0e00--0x0fff "TPU Channels"
   TEXTLINE ""
   TEXTLINE "-----------------------------------------------"
   TEXTLINE "CH FUNC PRIO HSF HSR IEF ISF LNK SGL CHS PRM0"
   TEXTLINE " 0,Channel 0"
   BITFLD.WORD 0x1e 0.--1. " " "Off,Low,Mid,High"
```

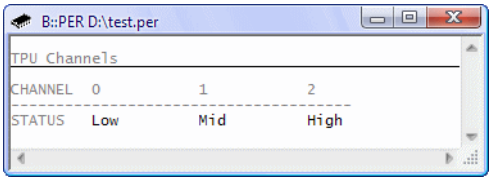


TEXTFLD Define text header

Format:	**TEXTFLD "***<text>***"**
<text>:	Optional text.

Defines text without creating a new line.

```
GROUP d:0x80000000--0x80000fff "TPU Channels"
TEXTLINE ""
TEXTLINE "CHANNEL "
TEXTFLD " 0,Channel 0"
TEXTFLD " 1,Channel 1"
TEXTFLD " 2,Channel 2"
TEXTLINE "--------------------------------------"
TEXTLINE "STATUS  ,Status"
BITFLD.WORD 0x0 0.--1. " " "Off,Low,Mid,High"
BITFLD.WORD 0x0 2.--3. " " "Off,Low,Mid,High"
BITFLD.WORD 0x0 4.--5. " " "Off,Low,Mid,High"
```



Functions

The table below shows an extract of fuctions usefull for writing PER files.

For a complete list of available functions please see:

- **IDE Functions**
- **General Functions**
- **STG Functions**

<int>	**CONV.INTTOBOOL(***<bool>***)**	Converts a boolean value to an integer. TRUE becomes 1, FALSE becomes 0 This function allows you to write conditional base statements e.g.: `base VM:(0x1010*conv.booltoint(d.l(vm:0)==42)|0x1070*conv.booltoint(d.l(vm:0)!=42)`
<int>	**PER.ARG(***<index>***)**	Returns the (optional) argument of the **Per.view** command. The parameter is currently not used. Only useful inside peripheral definition files.
<int> *<int>*	**PER.BUFFER.BYTE(***<index>***)** **PER.B.B(***<index>***)**	Returns a byte from the SGROUP buffer. Only usefull within a **SGROUP** of a PER-file.
<int> *<int>*	**PER.BUFFER.WORD(***<index>***)** **PER.B.W(***<index>***)**	Returns a 16 bit word from the SGROUP buffer. Only usefull within a **SGROUP** of a PER-file.
<int>	**PER.BUFFER.LONG(***<index>***)** **PER.B.L(***<index>***)**	Returns a 32 bit word from the SGROUP buffer. Only usefull within a **SGROUP** of a PER-file.
<int>	**PER.BUFFER.QUAD(***<index>***)** **PER.B.Q(***<index>***)**	Returns a 64 bit from the SGROUP buffer. Only usefull within a **SGROUP** of a PER-file.
<addr>	**PER.EVAL(***<index>***)**	Returns the value of a expression (defined with **BASE**) inside a peripheral definition file (PER file), which was defined after **BASE**, **IF**, **ELIF** or **ELSE** command. The parameter defines which expression is returned (0=first one). **Note:** The function returns only the last evaluated value of the expression. It will not evaluated the expression again. Expressions after **BASE**, will be evaluated by a **GROUP** command after the **BASE** command in a PER file.

Build 21955 27. Feb. 2010	New commands **BASEOUT** and **BASESAVEOUT**
Build 21439 28. Jan. 2010	New function **CONV.BOOLTOINT()**.
Build 21331 20. Jan. 2010	New command **ASSERT**.
Build 21299 19. Jan. 2010	New command **VARX** for **SGROUP**.
Build 20627 24. Nov. 2009	Enhanced **SGROUP** commands **SET** and **CONSTX** to accept also bitmasks and hexmasks for <*value*>.
Build 18839 28. Jul. 2009	New command **DECMASC**, to show decimal numbers.
Build 13442 28. Mai. 2008	PER programming enhanced to support nested IFs