New and Updated Functions since Tiger BASIC Version 5.2 manual

Blank Page

Index

Blank Page

Tiger-BASIC Preprocessor Instructions

#comment, #endcomment, #cm, #endcm

Available from version 4.6n

```
#comment        (or #cm)
...
#endcomment     (or #endcm)
```

Function: Marks the start respective the end of a comment area in the source text. Comments can also be nested.

The example shows the short form of the same commands:

```
#cm
  PRINT #1, "ABC"
#endcm
```

An example for nested comments:

```
#CM                     ' start of comment (a)
PRINT #1, "abc"         ' not compiled, this is comment
#CM                     ' start of comment (b)
WAIT_DURATION 1000      ' not compiled, this is comment
#ENCDM                  ' end of comment (b)
PUT #2, "c"             ' not compiled, this is comment
#ENCDM                  ' end of comment (a)
```

#define, #undef

#define available from version 1.2
#undef available from version 5.4

```
#define Text1 Text2

#undef Text1
```

Function: #define substitutes all Text1 in the following code with Text2. Text1 ends with the first space character or tab character. Text1 is not substituted if in quotation marks or in connection with other letters. Text2 may also include space and tab characters and ends with the line end or comment sign. When used in an #ifdef instruction Text1 from this point on will give a true as result.
#undef ends the substitution of Text1 for the following code. When used in an #ifdef instruction Text1 from this point on will give a false as result.

Example for defines:

```
#define LCD 1                    ' all text "LCD" is replaced by "1"
#define CLS CALL CLEAR_LCD       ' "CLS" is replaced by "CALL CLEAR_LCD"
#define RINT 2                   ' "RINT" -> "2" but not "PRINT" -> "P2"
```

Example for define / undef:

```
#define ErrCode 35       ' "ErrCode" is replaced by "35"
PRINT #1, ErrCode        ' The text "35" is printed
#undef ErrCode           ' "ErrCode" is no longer replaced
PRINT #1, ErrCode        ' "ErrCode" is now interpreted as variable and its
                         ' value is printed
```

#define TIGER_1 or #define TIGER_2

Available from version 5.3

#define TIGER_1 ***(or #define TIGER_2)***

Function: The symbolic constants "TIGER_1" and "TIGER_2" are automatically generated by compiler and can be applied for managing the module-dependent branches of the source code. Creating these defined in your code may result in unwanted effects running your program and should thus be avoided.

Example for installing serial driver with different baud rates using Tiger 1 or Tiger 2:

```
#ifdef TIGER_1
  INSTALL_DEVICE #SER, "SER1B_K1.TDD", &
    BD_38_400, DP_8N, JA, &                    ' settings for SER0
    BD_38_400, DP_8N, JA                       ' settings for SER1
#else
  INSTALL_DEVICE #SER, "SER1B_K1.TD2", &
    BD_115_200, DP_8N, JA, &                   ' settings for SER0
    BD_115_200, DP_8N, JA                      ' settings for SER1
#endif
```

#ifdef, #ifndef, #elif, #else, #endif

Available from version 5.01a

```
#ifdef          (or #ifndef)
...
#else           (or #elif)
...
#endif
```

Function: Conditional compilation (in simplest form). Any word, that occurs on the left side of the #define instruction, is meant to be defined independently of "assigned" value and will be evaluated as true by #ifdef, #ifndef and #elif instructions.

Example:

```
#define DEBUG
…
#ifdef DEBUG
  PRINT #LCD, DebugValue         ' output only if DEBUG is defined
#endif
```

#project_path, #project_path_dyn

Available from version 5.3

#project_path "[path]"

#project_path_dyn "[path]"

Function: Paths to particular files (Includes, bitmaps, data etc.) the project is built from can be set directly in the source code.

- The "#project_path" can either be an absolute path or a relative path from the path the TIG file of the project is located.
- The "#project_path_dyn" is a relative path added to all paths set with the "#project_path" instruction.

Several project paths and dynamical project paths may be set in a project.

Example:

```
#project_path "..\Definitions;..\IncludeFiles;"
#project_path_dyn "Sounds;Graphics;"
```

Assuming your project's TIG file is located in path "..\MyProjects\Project01", instruction #project_path would add the paths "..\MyProjects\Definitions" and "..\MyProjects\IncludeFiles" to look for project files.

Additional to these, instruction #project_path_dyn adds the directories
- "..\MyProjects\Definitions\Sounds",
- "..\MyProjects\Definitions\Graphics",
- "..\MyProjects\IncludeFiles\Sounds" and
- "..\MyProjects\IncludeFiles\Graphics"

to the list of searched paths.

#project_model

Available from version 4.6n (PM_FULL, PM_MIN)
Available from version 5.4 (PM_MEDIUM, PM_SMALL_W)

#project_model *model_name*

Function: With newer versions the Tiger-BASIC™ run time kernel is increasing in size as it contains more and more functions and instructions. Thus applications made for a certain module may get too large when compiled with a new run time kernel. With #project_model you can choose between the full run time kernel and smaller versions. The following project models are available:

- **PM_FULL**: By default the compiler uses the full model containing all functions and instructions.
- **PM_MEDIUM**: All functions and instructions are available except for all signal- and graphical functions.
- **PM_SMALL_W2**: Contains all functions and instructions of PM_MEDIUM except for X-Port functions, ASC, BCD, PBCD, Bit Modify String, ANIB, INIB, Shift Stri, Scramble, Prime, Serial No, byte & bit mix & demix, I²C functions, I²CL functions, HDQ, 1-wire, FPGA, KEY_DIRECT, LOOKUP functions.
- **PM_SMALL_W**: Contains all functions and instructions of PM_SMALL_W2 except for the PROGHASH function.
- **PM_MIN**: The minimum compiler model does not support instructions and functions introduced since version 3.0. In the “Tiger-BASIC v5.0 Programming Manual”, section “Annotated Overview”, only instructions and functions included in the full project model are marked. The same identification can also be found on every page of the detailed description. Only instructions and function unmarked there are available in this model.

Example:

```
#project_model pm_medium
```

New functions

CALIBRATE_LONG

RES = CALIBRATE_LONG (value, start, adjustment, interval)

Function: Adjusts a LONG value (e.g. from Real Time Clock).

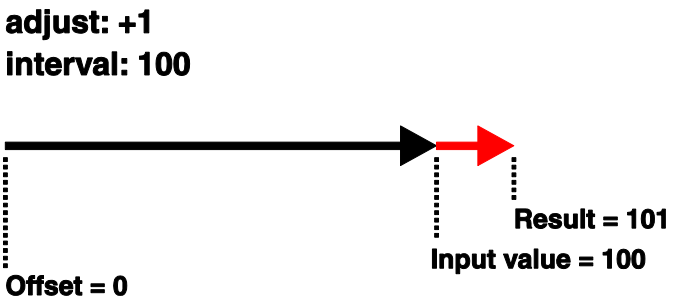
Parameters:

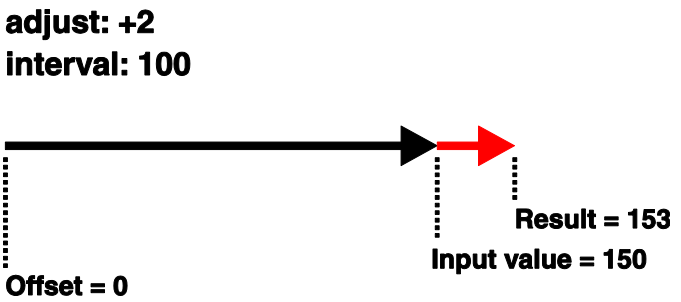
	B	W	L	S	F	
value	●	●	●	-	-	**Input** value to adjust
start	●	●	●	-	-	Starting point, from here adjustment begins. Values smaller than *start* are NOT adjusted.
adjustment	●	●	●	-	-	Adjustment value
interval	●	●	●	-	-	Interval over which the adjustment value is added. The added value is given by (interval / adjustment), thus intermediate steps are regarded.
						Function value:
RES	●	●	●	-	-	calibrated value

The function CALIBRATE_LONG adjusts a long counter, e.g. a real time clock. For every *interval*, *adjustment* is added to the input value. Intermediate steps are regarded, so e.g. an *adjustment* of 2 with an *interval* of 100 will lead to the same result as an *adjustment* of 1 with an *interval* of 50. The adjustment begins at the *start* value, values below the *start* value are NOT adjusted.

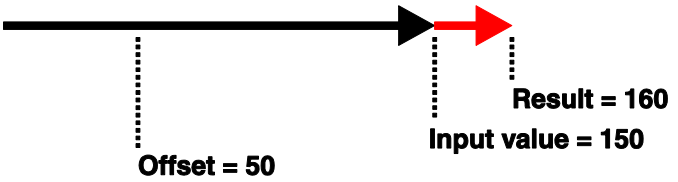
Example(*start* = 100, *interval* = 100, *adjustment* = 2)

Input value	Adjusted value	Comment
0	0	Value NOT adjusted
50	50	Value NOT adjusted
100	100	Here adjustment starts
150	151	Intermediate step
200	202	First full interval

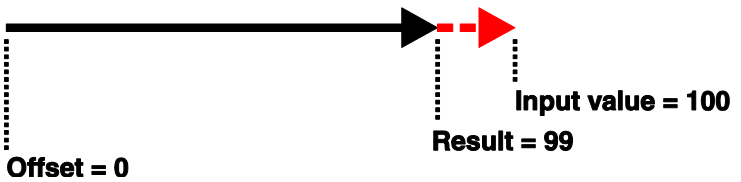




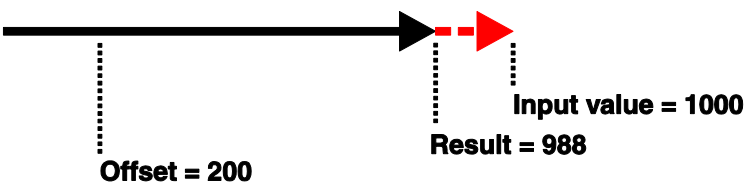
adjust: +5
interval: 50



adjust: -1
interval: 100



adjust: -3
interval: 200



Example:

```
RES = CALIBRATE_LONG(200, 100, 1, 100)
```

Example(RTC runs 2 seconds too fast per day):

```
user_var_strict
#INCLUDE UFUNC4.INC                       ' User Function Codes
#INCLUDE DEFINE_A.INC                     '


TASK Main                                 ' Beginn Task MAIN
  LONG Seconds, Prev_Sec                  ' LONG-Variablen deklarieren
  LONG calibratedSeconds
  BYTE RTCSTAT
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL DEVICE #3, "RTC1.TDD"           ' install device driver

  RTCSTAT = RTC_INITIAL                   '
  WHILE RTCSTAT < RTC_NO_RTC              ' search RTC
    GET #3, #0, #UFCI_RTC_STAT0, 1, RTCSTAT ' get status of RTC
    PRINT #1,"<1>installing";             '
    WAIT_DURATION 200                     '
  ENDWHILE                                '
  IF RTCSTAT = RTC_PRESENT THEN           ' if RTC availiable
    Seconds = 12345678                    ' initial value
    PUT #3, Seconds                       ' set RTC time (seconds)
    RTCSTAT = RTC_BUSY                    '
    WHILE RTCSTAT = RTC_BUSY              ' while RTC is buse
      GET #3, #0, #UFCI_RTC_STAT1, 1, RTCSTAT ' read status of RTC
      PRINT #1,"<1>busy";                 '
      WAIT_DURATION 200                   '
    ENDWHILE                              '
    LOOP 9999999                          ' many loops
      Prev_Sec = Seconds                  ' save old time
      WHILE Seconds = Prev_Sec            ' while time has not changed
        GET #3,0, Seconds                 ' read out time from RTC
      ENDWHILE                            '
      PRINT #1,"<1>RTC-Time =<0>";Seconds ' show time from RTC

      ' calibrate time from RTC
      calibratedSeconds = CALIBRATE_LONG(Seconds, &     ' Input variable
      12345678, &               ' start (RTC was set to this initial value)
      -2, &                     ' adjustment (2 seconds too fast per day)
      86400)                    ' interval (1 Day = 86400 seconds)

      PRINT #1,   "Cal-Time =<0>";calibratedSeconds ' show calibrated time
    ENDLOOP                               '
  ELSE                                    ' no RTC
    PRINT #1, "<1>No RTC found"           '
  ENDIF                                   '
END                                       '
```

DISABLE_TSW_NESTED

DISABLE_TSW_NESTED()

Function: Switches Task Switching mechanism OFF recursive.

ENABLE_TSW_NESTED inhibits task-switching, places the complete CPU performance at the disposal of the current task and an internal counter is incremented. This function is used in combination with *DISABLE_TSW_NESTED.*

The *TSW_NESTED* functions are ideal for subroutines or greater programs like the Tiger Graphic Library. If there are time critical phases in subroutines or global variables have to be manipulated undisturbed, these subroutines can be called anywhere in the program. *ENABLE_TSW* will enable the Task Switching unconditional, but after calling *DISABLE_TSW_NESTED* two times, *ENABLE_TSW_NESTED* also must be called two times to enable the Task Switching again. *RELEASE_TSW_NESTED* operates only, if the counter is 0. The maximum value of the internal counter is 255.

Also see: ENABLE_TSW_NESTED, RESET_TSW_NESTED, READ_TSW_NESTED and RELEASE_TSW_NESTED.

Example:

```
'------------------------------------------------------------------------------
' This program demonstrates the difference between
' disable_tsw_nested(), enable_tsw_nested()
' and
' disable_tsw, enable_tsw
'
' After calling the subroutine, enable_tsw will enable the Task Switching
' unconditional, but the first disable_tsw_nested() in this case will not
' activate the Task Switching.
'------------------------------------------------------------------------------
user_var_strict

task main
  run_task blink

  disable_tsw_nested()
  call nested
  ' Task Switching is disabled here
  wait_duration 5000
  enable_tsw_nested()

  ' impossible with standard instructions
  disable_tsw
  call standard
  ' Task Switching is enabled here ...
  wait_duration 5000
  enable_tsw

end

sub nested
  disable_tsw_nested()
  ' your code here
  enable_tsw_nested()
end

sub standard
  disable_tsw
  ' your code here
  enable_tsw ' enables tsw unconditional
end

task blink
 DIR_PORT 8, 0          ' Port-8 Output

 while 1=1              ' Loop
   OUT 8, 255, 255      ' LEDs on
   wait_duration 1000   ' wait 1 sec
   OUT 8, 255, 0        ' LEDs off
   wait_duration 1000   ' wait 1 sec
 endwhile               ' Endloop

end
```

ENABLE_TSW_NESTED

ENABLE_TSW_NESTED()

Function: Switches Task Switching mechanism ON recursive.

ENABLE_TSW_NESTED decrements an internal counter. If the counter is greater than zero, nothing else happens, otherwise the Task Switching is enabled, the counter is set to 0 and all tasks continue according to their priority. This function is used in combination with *ENABLE_TSW_NESTED.*

The *TSW_NESTED* functions are ideal for subroutines or greater programs like the Tiger Graphic Library. If there are time critical phases in subroutines or global variables have to be manipulated undisturbed, these subroutines can be called anywhere in the program. *ENABLE_TSW* will enable the Task Switching unconditional, but after calling *DISABLE_TSW_NESTED* two times, *ENABLE_TSW_NESTED* also must be called two times to enable the Task Switching again. *RELEASE_TSW_NESTED* operates only, if the counter is 0. The maximum value of the internal counter is 255.

Also see: DISABLE_TSW_NESTED, RESET_TSW_NESTED, READ_TSW_NESTED and RELEASE_TSW_NESTED.

Example:

```
user_var_strict

task main

  run_task blink

  ' Port-8 will blink 5 seconds
  wait_duration 5000


  disable_tsw_nested()
  disable_tsw_nested()
  disable_tsw_nested()
  wait_duration 4000    ' no blinking
  enable_tsw_nested()
  wait_duration 4000    ' no blinking
  enable_tsw_nested()
  wait_duration 4000    ' no blinking
  enable_tsw_nested()
  wait_duration 4000    ' blink again

  disable_tsw_nested()  '
  disable_tsw_nested()  '
  disable_tsw_nested()  '
  reset_tsw_nested()    ' reset internal counter
  enable_tsw_nested()   ' blink again

end


task blink

 DIR_PORT 8, 0          ' Port-8 Output

 while 1=1              ' Loop
   OUT 8, 255, 255      ' LEDs on
   wait_duration 1000   ' wait 1 sec
   OUT 8, 255, 0        ' LEDs off
   wait_duration 1000   ' wait 1 sec
 endwhile               ' Endloop

end
```

FPGA_PROGRAM

**Flag = FPGA_PROGRAM (FPGA_Bin_File,&
Len)**

Function: Writes a FPGA binary file.

Parameters:

	B	W	L	S	F	
FPGA_Bin_File	●	●	●	-	-	Flash-ADDR of FPGA binary file.
Len	●	●	●	-	-	Length of FPGA binary file.
						Function value:
Flag	●	●	●	-	-	0 = OK, all Bytes written successfully 1 = Flash-ADDR not valid 2 = Len not valid 3 = Timeout (no device connected) 4 = Please call FPGA_SETUP first

FPGA_PROGRAM writes a compiled binary file to XILINX Spartan-II 2,5V FPGA Family controllers. Ensure to perform FPGA_SETUP once before. The transfer rate of the Tiger-1 is about 6.5µs / Byte.

Example:

```
user_var_strict

task main
  datalabel FPGA_BIN_FILE, FPGA_BIN_FILE_END
  long llResult
  long llFileLen



  ' init the FPGA function
  llResult = FPGA_SETUP( &
  6, &  ' Databus
  7, &  ' /CS Port
  3, &  ' /CS Pin No.
  8, &  ' CTRL Port
  0, &  ' CCLK Pin No.
  1, &  ' /INIT Pin No.
  6 &   ' /PROG Pin No.
  )

  ' get length of file
  llFileLen = FPGA_BIN_FILE_END - FPGA_BIN_FILE


  ' Program the FPGA controller
  llResult = FPGA_PROGRAM( &
  FPGA_BIN_FILE, &      ' Start of file
  llFileLen &           ' Length of file
  )

FPGA_BIN_FILE::
DATA FILE  "tdr1000_logic_v1_0.bit"
FPGA_BIN_FILE_END::
end
```

FPGA_SETUP

```
Flag = FPGA_SETUP (     Bus_Port,     &
                        CS_Port,      &
                        Bit_CS,       &
                        CTRL_Port,    &
                        Bit_CCLK,     &
                        Bit_INIT,     &
                        Bit_PROG)
```

Function: Specifies Tiger pins used for FPGA programming.

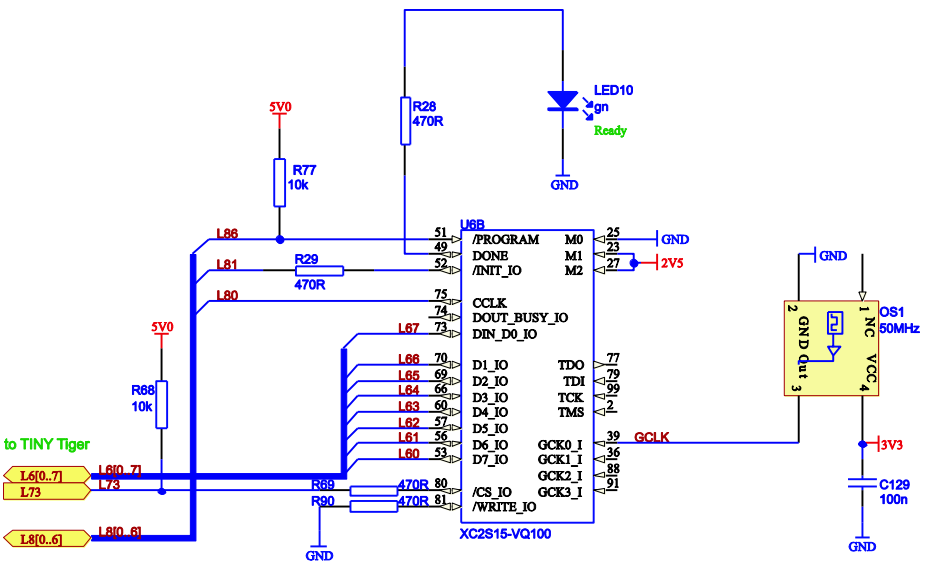
Parameters:

	B	W	L	S	F	
Bus_Port	●	●	●	-	-	Port, which is used as an 8-bit DATA bus.
CS_Port	●	●	●	-	-	Port, which is used for /CS.
Bit_CS	●	●	●	-	-	Bit-no: 0...7 for /CS.
CTRL_Port	●	●	●	-	-	Port, which is used for CCLK, /INIT and /PROG.
Bit_CCLK	●	●	●	-	-	Bit-no: 0...7 for CCLK.
Bit_INIT	●	●	●	-	-	Bit-no: 0...7 for /INIT
Bit_PROG	●	●	●	-	-	Bit-no: 0...7 for /PROG.
						Function value:
Flag	●	●	●	-	-	0 = OK, parameters accepted 1 = DATA bus does not exist 2 = /CS Port does not exist 3 = /CS Pin invalid 4 = /CS Pin already in use 5 = CTRL Port does not exist 6 = CCLK Pin invalid 8 = /PROGRAM Pin invalid 9 = /INIT Pin invalid 10 = One or more of the CTRL Pins are already in use.

FPGA_SETUP assigns Tiger I/O pins to the signals of the FPGA controller. It is not necessary to set the direction of the pins. The common procedure is to call FPGA_SETUP once and then you can use FPGA_PROGRAM every time. During the program phase of the FPGA, also other device drivers can use the bus, CCLK and /INIT. A running FPGA transmission is interrupted by such request if necessary. In this case /CS is set to “inactive” during this time and the FPGA will ignore all signals. Only /CS and /PROG cannot be used by other device drivers or functions.

The functions can be used for all **XILINX Spartan-II 2,5V FPGA Family** controllers in slave parallel mode. One of the Wilke products with a FPGA is the TDR-1000.

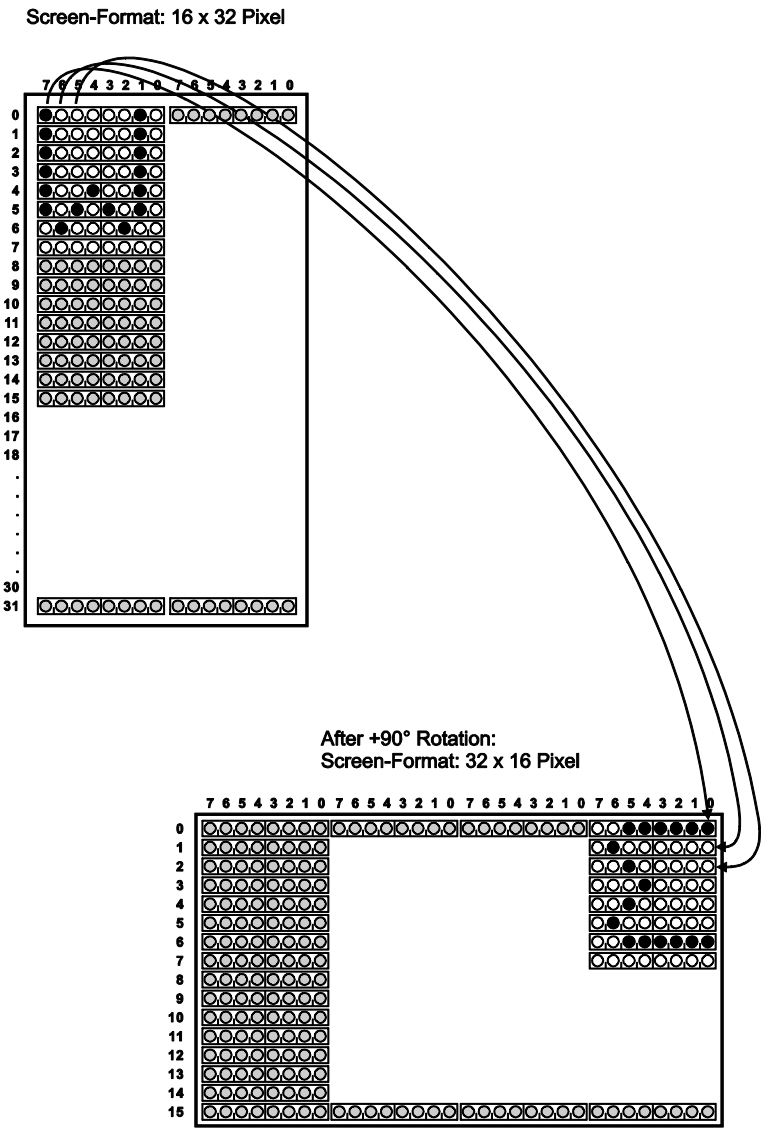
The /WRITE line of FPGA has to set to logic 0 by a 470 ohm pull down resistor. The DONE pin could be connected to a LED over a 470 ohm resistor to signal the state of the FPGA. If the LED lights on, FPGA program successfully loaded and it is ready to use. **Ensure to set the /PROG line of FPGA to logic 1 by a 10 kohm pull up resistor.** Please protect the /INIT and /CS lines of the FPGA with 470 ohm serial resistors. Please use the bus lines D0...D7 only as data bus input.



GRAPHIC_ROTATE

Graphic_Rotate (Src, Destin$, Width, Height)

Function: Rotates a graphic 90° to the right.



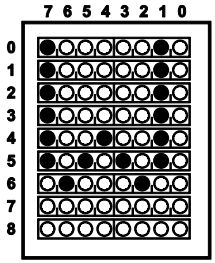
Parameters:

	B	W	L	S	F	
Src	●	●	●	●	-	Source graphic (STRING or FLASH address, e.g. Datalabel)
Destin$	-	-	-	●	-	Destination with rotated pixels
Width	●	●	●	-	-	Format width in pixels: 1 ... nnnn
Height	●	●	●	-	-	Format height in pixels: 1 ... nnnn

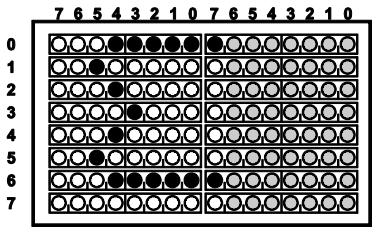
Src$ and Destin$ must not be the same string

The source graphic must have a width multiple of 8, but GRAPHIC_ROTATE processes all widths and heights, even if they are not a multiple of 8. In this case the length of source and destination can vary:

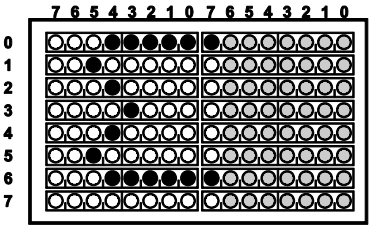
Screen-Format: 8 x 9 Pixel



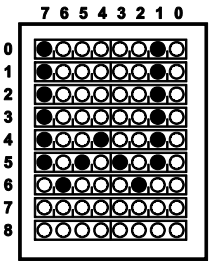
After +90° Rotation:
Screen-Format: 9 x 8 Pixel



Screen-Format: 9 x 8 Pixel



After +90° Rotation:
Screen-Format: 8 x 9 Pixel



The fastest way to rotate a graphic 180° is to mirror the graphic with the function **GRAPHIC_MIRROR.** A rotation of 270° to the right or 90° to the left can be processed with a mirror (180°) and one rotation (90°).

Example of 90° rotation:

```
graphic_rotate(Src$, Dst$, width, height)
```

Example of 180° rotation:

```
graphic_mirror(Src_Dst$, width, height, 3)
```

Example of 270° rotation:

```
graphic_rotate( Src$, Dst$, width, height )
newWidth = (height + 111b) BITAND 0FFFFFFF8H
newHeight = width
graphic_mirror( Dst$, newWidth, newHeight, 3 )
```

PROGHASH

RES = PROGHASH (Start, Length)

Function: Calculates 32-Bit hash code from BASIC program in specified range.

Parameters:

	B	W	L	S	F	
Start	●	●	●	-	-	Start address in BASIC program, the address 0 is the first BASIC instruction.
Length	●	●	●	-	-	Length 0: all Bytes, to the end of the BASIC program
						Function value:
RES	●	●	●	-	-	32-bit hash code 0: invalid parameters

The PROGHASH function calculates a 32 bit hash code over the **BASIC program** (**no** user flash area, drivers or TAC files). The hash code of exactly the same BASIC program equals on every Tiger. Data in the user flash does not have any influence to the hash code.

To calculate the hash code over the complete BASIC program, set both parameters, start address and length to 0:

```
hashcode = PROGHASH( 0, 0 )
```

You can use the PROGHASH function to identify exactly your program, e.g. by sending the hash code to an external system. If you vary the input parameters, the returned hash code is not predictable. You can read out the length of the BASIC program with SYSVARN:

```
len = SYSVARN( BAS_PROG_LEN, 0 )
```

PROGHASH requires Tiger BASIC IDE 5.4.4 or higher!

READ_TSW_NESTED

RES = READ_TSW_NESTED()

Function: Reads out counter of the *TSW_NESTED* functions.

Parameters:

	B	W	L	S	F	Function value:
RES	●	●	●	-	-	counter value

The *TSW_NESTED* functions are ideal for subroutines or greater programs like the Tiger Graphic Library. If there are time critical phases in subroutines or global variables have to be manipulated undisturbed, these subroutines can be called anywhere in the program. *ENABLE_TSW* will enable the Task Switching unconditional, but after calling *DISABLE_TSW_NESTED* two times, *ENABLE_TSW_NESTED* also must be called two times to enable the Task Switching again. *RELEASE_TSW_NESTED* operates only, if the counter is 0. The maximum value of the internal counter is 255.

Also see: ENABLE_TSW_NESTED, DISABLE_TSW_NESTED, RESET_TSW_NESTED and RELEASE_TSW_NESTED.

RELEASE_TSW_NESTED

RELEASE_TSW_NESTED()

Function: Releases the CPU time allocation for a particular task recursive.

RELEASE_TSW_NESTED checks the counter of the *TSW_NESTED* functions. If the counter is greater than zero, nothing happens, otherwise the CPU time is released for other tasks.

This instruction enables a different form of dynamic prioritizing and dynamic distribution of CPU resources, throughout the program execution.

Example: A certain job only occasionally has jobs to perform. However, when these occur they must be processed rapidly. The task thus runs with a high priority. If no job is pending, a *RELEASE_TSW_NESTED* instruction is executed to immediately release the CPU time for other tasks, if no recursive task switching is disabled.

The *TSW_NESTED* functions are ideal for subroutines or greater programs like the Tiger Graphic Library. If there are time critical phases in subroutines or global variables have to be manipulated undisturbed, these subroutines can be called anywhere in the program. *ENABLE_TSW* will enable the Task Switching unconditional, but after calling *DISABLE_TSW_NESTED* two times, *ENABLE_TSW_NESTED* also must be called two times to enable the Task Switching again. *RELEASE_TSW_NESTED* operates only, if the counter is 0. The maximum value of the internal counter is 255.

Also see: ENABLE_TSW_NESTED, DISABLE_TSW_NESTED, RESET_TSW_NESTED and READ_TSW_NESTED.

RESET_TSW_NESTED

RESET_TSW_NESTED()

Function: Resets counter for *ENABLE_TSW_NESTED* and *DISABLE_TSW_NESTED*.

The internal counter for *ENABLE_TSW_NESTED* and *DISABLE_TSW_NESTED* is set to 0.

The *TSW_NESTED* functions are ideal for subroutines or greater programs like the Tiger Graphic Library. If there are time critical phases in subroutines or global variables have to be manipulated undisturbed, these subroutines can be called anywhere in the program. *ENABLE_TSW* will enable the Task Switching unconditional, but after calling *DISABLE_TSW_NESTED* two times, *ENABLE_TSW_NESTED* also must be called two times to enable the Task Switching again. *RELEASE_TSW_NESTED* operates only, if the counter is 0. The maximum value of the internal counter is 255.

Also see: ENABLE_TSW_NESTED, DISABLE_TSW_NESTED, READ_TSW_NESTED and RELEASE_TSW_NESTED.

SCALE

RES = SCALE (value, numerator, denominator)

Function: The function SCALE adjusts the input value in a proportional ratio by multiplying it with the quotient of numerator and denominator.

Parameters:

	B	W	L	S	F	
value	●	●	●	-	-	Input value to adjust Minimum vale: -32768 Maximum value: 32767
numerator	●	●	●	-	-	Scale factor Minimum vale: 0 Maximum value: 32767
denominator	●	●	●	-	-	Corresponds to 100% Minimum vale: 10 Maximum value: 32767
						Function value:
RES	●	●	●	-	-	Scaled value.

A usual value for the denominator is for example 1.000. If you use a denominator of 100, the function SCALE calculates the percentage.

Please notice the minimal and maximal values of the parameters. If an input value exceed the corresponding limitations, the value is set to the minimum or maximum.

Example: You want to know what 35% of 80 is. The input value is 80, the numerator 35 and the denominator 100. The result of this operation is 28.

```
RES = SCALE( 80, 35, 100 )      ' RES = 28
```

Example:

```
user_var_strict
#include DEFINE_A.inc

TASK MAIN                                 'begin task MAIN
  LONG RES, VALUE
'install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
'install LCD-driver (TINY-Tiger)
'INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  PRINT #LCD, "SCALE.TIG"

  VALUE = 80                              '
  RES = SCALE( VALUE, 35, 100 )           '
  PRINT #LCD, "SCALE(80,35,100)=";RES;  ' result: 28

  VALUE = 100                             '
  RES = SCALE( VALUE, 50, 1000 )          '
  PRINT #LCD, "SCALE(80,35,100)=";RES   ' result: 5
END                                       'end task MAIN
```

UNINSTALL_DEVICE

uninstall_device #Dev_No, Dummy

Function: uninstalls a **TimerA** dependant device driver.

Parameters:

	B	W	L	S	F	
#Dev_No	●	●	●	-	-	is a constant in this instruction and specifies the number of the I/O device.
Dummy	●	●	●	●	●	Dummy value. Set to 0!

The job of the device drivers is to manage device specific operations independently. You can install 8 TimerA dependant device drivers at the same time. TimerA device drivers can cause a heavy CPU load. To decrease the CPU load of the TimerA device drivers and free CPU load for the BASIC program or to alternate more than 8 TimerA device drivers with this instruction it is possible to uninstall one device driver at a time.

It is possible to install a device driver again after uninstalling. To reinstall a device driver, just use the *install_device* with the same *#Dev_No*. Please use one unique *#Dev_No* for each driver; never use the same *#Dev_No* for more than one device driver. You can now use the *install_device* everywhere in the program, not just in the Task Main. You need the compiler version (IDE) 5.3.1 or above.

After a new installation, all previous settings are lost. It could be helpful to put the installation and configuration of a device driver to a subroutine.

Example of uninstall_device:

```
user_var_strict

task main
  INSTALL_DEVICE #0, "TIMERA.TDD",2,173 ' 3612Hz => 1200baud
  INSTALL_DEVICE #2,"SER2_74_R1.TD2", &
    8, &  ' data bits
    0, &  ' parity 0=no parity
    0, &  ' invert 0=true, 1=invers
    3, &  ' tx Prescaler
    3, &  ' rx Oversample
    1, &  ' reserved, always 1
    0     ' handshake, 0=no handshake
  print #2, "SER2_74_R1.TD2"
  wait_duration 500
  uninstall_device #2, 0                ' uninstall SER2_74_R1.TD2

  INSTALL_DEVICE #2,"SER2_74_R1.TD2", & ' install SER2_74_R1.TD2 again
    8, &  ' data bits
    0, &  ' parity 0=no parity
    0, &  ' invert 0=true, 1=invers
    3, &  ' tx Prescaler
    3, &  ' rx Oversample
    1, &  ' reserved, always 1
    0     ' handshake, 0=no handshake
  print #2, "SER2_74_R1.TD2 installed again"
end
```

Example of many uninstall_device:

```
#include define_a.inc
user_var_strict
user_eport act, noactive


task main
  dir_pin 8, 4, 0

  INSTALL_DEVICE #0, "TIMERA.TDD",2,173          ' 3612Hz => 1200baud
  install_device #1, "SET1.TD2", 84              ' monitor on L84
  INSTALL_DEVICE #2,"SER2_60_R1.TD2", 8,0,0,0,3,1,0' install device drivers
  INSTALL_DEVICE #3,"SER2_61_R1.TD2", 8,0,0,0,3,1,0' install device drivers
  INSTALL_DEVICE #4,"SER2_62_R1.TD2", 8,0,0,0,3,1,0' install device drivers
  INSTALL_DEVICE #5,"SER2_63_R1.TD2", 8,0,0,0,3,1,0' install device drivers
  INSTALL_DEVICE #6,"SER2_64_R1.TD2", 8,0,0,0,3,1,0' install device drivers
  INSTALL_DEVICE #7,"SER2_65_R1.TD2", 8,0,0,0,3,1,0' install device drivers
  install_device #50, "RES1.TD2", 84             ' monitor on L84

  uninstall_device #2, 0                         ' uninstall device driver
  uninstall_device #3, 0                         ' uninstall device driver
  uninstall_device #4, 0                         ' uninstall device driver
  uninstall_device #5, 0                         ' uninstall device driver
  uninstall_device #6, 0                         ' uninstall device driver
  uninstall_device #7, 0                         ' uninstall device driver


  INSTALL_DEVICE #8,"SER2_60_K1.TD2", 8,0,0,0,3,1,0 ' install other drivers
  INSTALL_DEVICE #9,"SER2_61_K1.TD2", 8,0,0,0,3,1,0 ' install other drivers
  INSTALL_DEVICE #10,"SER2_62_K1.TD2", 8,0,0,0,3,1,0' install other drivers
  INSTALL_DEVICE #11,"SER2_63_K1.TD2", 8,0,0,0,3,1,0' install other drivers
  INSTALL_DEVICE #12,"SER2_64_K1.TD2", 8,0,0,0,3,1,0' install other drivers
  INSTALL_DEVICE #13,"SER2_65_K1.TD2", 8,0,0,0,3,1,0' install other drivers


  uninstall_device #8, 0                         ' uninstall device driver
  uninstall_device #9, 0                         ' uninstall device driver
  uninstall_device #10, 0                        ' uninstall device driver
  uninstall_device #11, 0                        ' uninstall device driver
  uninstall_device #12, 0                        ' uninstall device driver
  uninstall_device #13, 0                        ' uninstall device driver


  INSTALL_DEVICE #2,"SER2_60_R1.TD2", 8,0,0,0,3,1,0' first drivers again
  INSTALL_DEVICE #3,"SER2_61_R1.TD2", 8,0,0,0,3,1,0' first drivers again
  INSTALL_DEVICE #4,"SER2_62_R1.TD2", 8,0,0,0,3,1,0' first drivers again
  INSTALL_DEVICE #5,"SER2_63_R1.TD2", 8,0,0,0,3,1,0' first drivers again
  INSTALL_DEVICE #6,"SER2_64_R1.TD2", 8,0,0,0,3,1,0' first drivers again
  INSTALL_DEVICE #7,"SER2_65_R1.TD2", 8,0,0,0,3,1,0' first drivers again
end
```

UPDATE_ME_FILE_INF$

RES = UPDATE_ME_FILE_INF$ (Data_Label, Option)

Function: Reads out information from an UPDATE_ME tgc/t2c file.

Parameters:

	B	W	L	S	F	
Data_Label	●	●	●	-	-	States the FLASH address, where a new version of the program exists in the Tiger DATA-FLASH area: 0 ... nnnn
Option	●	●	●	-	-	1 (*TGC_DATE*) = date & time (Result 17 Bytes: **"dd.mm.yy hh:mm:ss"**) 2 (*TGC_NAME*) = program name & path (Result 128 Bytes) 3 (*TGC_RAM_SIZE*) = RAM size 4 (*TGC_LEN*) = file length 5 (*TGC_COMP_VERS*) = compiler version (Result max. 6 Bytes: e.g. "5.4.1")
						Function value:
RES	-	-	-	●	-	Error Codes: "" = maximum length of string is too short to show result. "-1" =invalid flash address "-2" = fatal internal error "-3" = invalid file length (to small) "-4" = invalid file length (to large) "-5" = invalid flag "-6" = CRC-1 error "-7" = CRC-2 error "-8" = CRC-3 error "-9" = invalid Option "1" = program name not found

The result of the operation is always written in ASCII plain text. The result string can be shown on LCD directly.

```
updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_DATE )
print #LCD, updateMeFileInfo$
```

In case of *RAM size* and *file length*, it could make sense to save the result into a numeric variable:

```
updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_RAM_SIZE )
llRamSize = val_num(updateMeFileInfo$)
USING "UH<8><8> k 0 0 0 0 8"
print_using #LCD, "<1>"; llRamSize;
```

You can detect an error through the length of the result string. If the length of the result is less or equal 2, this must be an error code. An error occurs, if at the specified address is no valid update program.

```
updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_DATE )
if len(updateMeFileInfo$) <= 2 then
  llErrCode = val_num(updateMeFileInfo$)' error
endif
```

Program example:

```
USER_VAR_STRICT                          ' force declaration of variables
#include Define_a.inc                    ' general definitions

TASK MAIN                                ' begin task MAIN
  DATALABEL NEW_PROG_HERE                ' declare datalabel
  string updateMeFileInfo$(256)
  long llRamSize
  long llFileLen
  long llErrCode

  INSTALL_DEVICE #LCD, "LCD1.TD2"               ' install LCD-driver

'-------------------------------------------------------------------------
' date & time
'-------------------------------------------------------------------------
  updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_DATE )
  if len(updateMeFileInfo$) <= 2 then
    ' error
  endif
  print #LCD, "<1>"; updateMeFileInfo$
  wait_duration 2000

'-------------------------------------------------------------------------
' program name
'-------------------------------------------------------------------------
  updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_NAME )
  if len(updateMeFileInfo$) <= 2 then
    ' error
  endif
  updateMeFileInfo$ = trim$(updateMeFileInfo$," ", 0)
  print #LCD, "<1>"; updateMeFileInfo$;
  wait_duration 2000

'-------------------------------------------------------------------------
' RAM size
'-------------------------------------------------------------------------
  updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_RAM_SIZE )
  if len(updateMeFileInfo$) <= 2 then
    ' error
  endif
  llRamSize = val_num(updateMeFileInfo$)
  USING "UH<8><8> k 0 0 0 0 8"
  print_using #LCD, "<1>"; llRamSize;
  wait_duration 2000

'-------------------------------------------------------------------------
' file length
'-------------------------------------------------------------------------
  updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_LEN )
  if len(updateMeFileInfo$) <= 2 then
    ' error
  endif
  llFileLen = val_num(updateMeFileInfo$)
  print #LCD, "<1>"; llFileLen
  wait_duration 2000

'-------------------------------------------------------------------------
' compiler version
```

```
'-------------------------------------------------------------------------
  updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE, TGC_COMP_VERS )
  if len(updateMeFileInfo$) <= 2 then
    ' error
  endif
  print #LCD, "<1>"; updateMeFileInfo$
  wait_duration 2000

'-------------------------------------------------------------------------
' error
'-------------------------------------------------------------------------
  updateMeFileInfo$ = UPDATE_ME_FILE_INF$ ( NEW_PROG_HERE+1, 1 )
  if len(updateMeFileInfo$) <= 2 then
    llErrCode = val_num(updateMeFileInfo$)' error
  endif
  print #LCD, "<1>"; updateMeFileInfo$
  wait_duration 2000

'-------------------------------------------------------------------------
' new program
'-------------------------------------------------------------------------
NEW_PROG_HERE:: ' important: new file-type ".tgc", also see manual
#ifdef TIGER_1
  DATA FILE "UPDATE_ME_NEXT_GENERATION.tgc"' save compiled program code in
                                           ' FLASH, this will be next program
#endif

#ifdef TIGER_2
  DATA FILE "UPDATE_ME_NEXT_GENERATION.t2c"' save compiled program code in
                                           ' FLASH, this will be next program
#endif

END                                        ' end of task MAIN
```

VCDIFF_DECODE$

RES = VCDIFF_DECODE$ (Old$, Delta$, Buffer$, DecodedNew$, Winsize)

Restores DecodedNew$ from the original data in Old$ and the differences from it in Delta$. VCDIFF_DECODE$ is typically used to decompress and restore the reduced data size after transmission.

Parameters:

	B	W	L	S	F	
Old$	-	-	-	●	-	The initial source String. Differences are based on this String. This String must also be known for encoding.
Delta$	-	-	-	●	-	Differences between Old$ and DecodedNew$. Can be created with VCDIFF_ENCODE$.
Buffer$	-	-	-	●	-	Temporary buffer for internal operations of the algorithm.
DecodedNew$	-	-	-	●	-	Result with decoded data from the original data in Old$ and the differences from it in Delta$.
Winsize	●	●	●	-	-	Maximum chunk size. Needs space (Buffer$) proportionate to Winsize. Must be a power of 2. Can be adjusted to optimize the output size. 0: use default minimal Winsize (2048)
						Function value:
RES	-	-	●	-	-	Result / error Code of operation. For details, please refer to VCDIFF_ENCODE$

The Tiger supports VCDIFF_DECODE$ from TAP version 1.03g or higher!

More details about VCDIFF, the description of the result/error codes or example programs can be found at VCDIFF_ENCODE$!

Also see: VCDIFF_ENCODE$

VCDIFF_ENCODE$

RES = VCDIFF_ENCODE$ (Old$, New$, Buffer$, Delta$, Winsize)

Stores the differences between Old$ and New$ in Delta$. VCDIFF_ENCODE$ is typically used to compress and reduce the data size for transmission.

Parameters:

	B	W	L	S	F	
Old$	-	-	-	●	-	The initial source String. Differences are based on this String. This String must also be known for decoding.
New$	-	-	-	●	-	The next generation with changed data.
Buffer$	-	-	-	●	-	Temporary buffer for internal operations of the algorithm.
Delta$	-	-	-	●	-	Result with differences between Old$ and New$. Can be decoded with VCDIFF_DECODE$.
Winsize	●	●	●	-	-	Maximum chunk size. Needs space (Buffer$) proportionate to Winsize. Must be a power of 2. Can be adjusted to optimize the output size. 0: use default minimal Winsize (2048)
						Function value:
RES	-	-	●	-	-	0 = OK -1 = String Parameters must be unique -2 = Insufficient Delta$ output space -3, -4, -5, -6= unavailable secondary compressor (FGK, DJW, LZMA, unknown) -7 = Internal error -8 = Invalid Input (Encode: Old$, New$; Decode: Old$, Delta$) -9 = Out of memory. Increase Buffer$ size or decrease Winsize -17711 = Invalid config -17714 = Unimplemented feature

The Tiger supports VCDIFF_ENCODE$ from TAP version 1.03g or higher!

VCDIFF is a format and an algorithm for delta encoding. Delta encoding is a way of storing or transmitting data in the form of differences (deltas). In situations where differences are small – for example, the change of a few words in a large string or the change of a few records in a large table – delta encoding greatly reduces data redundancy.

For more details, please refer to the IETF's RFC 3284: https://datatracker.ietf.org/doc/html/rfc3284.

To encode or decode on the PC or any other machine, there are several free tools like Xdelta3: http://xdelta.org. For decoding on a Tiger please use VCDIFF_DECODE$.

Program example:

```
'--------------------------------------------------------------------------
' Name:  vcdiff.tig
'--------------------------------------------------------------------------
user_var_strict

task main
    datalabel dl_old, dl_new, dl_end
    string old$(4k)          ' must also be known when decoding
    string new$(4k)          ' the next generation with changed data
    string delta$(4k)        ' result with differences between old$ and new$
    string decodedNew$(4k)   ' result is equal to new$
    string buffer$(64k)      ' temporary RAM buffer is needed by vcdiff
    long ret

    peek_flash dl_old, old$, dl_new - dl_old
    peek_flash dl_new, new$, dl_end - dl_new

    'delta$ has a length of 81 bytes! (new$: 2585 bytes => ~97% compression)
    ret = vcdiff_encode$(old$, new$, buffer$, delta$, 0)
    ret = vcdiff_decode$(old$, delta$, buffer$, decodedNew$, 0)

dl_old::
    data file "initial-file.txt"
dl_new::
    data file "next-file.txt"
dl_end::
end
```

Also see: VCDIFF_DECODE$

XPort_4Samples

Flag = XPort_4Samples(MS_Count, Shift_Down, Buffer1$, ...,Buffer4$,& XPort1, ..., Xport4, Mask1, ...,Mask4)

Function: Samples in a given rate 4 xports and saves in case of changes the values of masked bits in a buffer for each xport.

Parameters:

	B	W	L	S	F	
MS_Count	●	●	●	-	-	Sample rate for xports (0...65535). ›=0: Starting values will NOT be written in buffer ‹0: Starting values will be written in buffer
Shift_Down	-	-	●	-	-	Number of bits to be downshifted of sampled values (0...7). This parameter contents 4 bytes, one for each xport. LSB stands for xport1. E.g. 01020304h means that the value of xport1 will be down shifted for 4 bits, xport2 for 3 bits, xport3 for 2 bits and xport1 for 1 bit
Buffer1$ ... 4$	-	-	●	-	-	Buffer for changed masked values on xports. Will be emptied with calling. No arrays of strings!
XPort1 ... 4	●	●	●	-	-	Physical address of sampled xport.(0...255)
Mask1 ... 4	●	●	●	-	-	Mask for sampled bits of xport. Changes of other bits will be ignored.
						Function value:
Flag	●	●	●	-	-	0: Ok, all parameters accepted ‹0: Number of one of invalid parameter

! Sampled xports can be additionally read with XIN and XPIN!

Arrays of strings are NOT accepted as string buffers!

Example:
Xports 0...3 will be sampled each 10ms. The changed values will be shifted down for 2 bits and saved in the string buffers S1$...S4$.

```
Flag = XPort_4Samples ( 10, 02020202h, &        ' sample rate, shift down
S1$,S2$,S3$,S4$, &                              ' buffers
0,1,2,3, &                                      ' xport addresses
00001100b, 00001100b, 00001100b, 00001100b )    ' masks
```

The sampling can be changed during the running program. To avoid undefined conditions, stop the sampling first. Just set the sample rate to zero.

Example:
Changing of number of down shifted bits and masks.

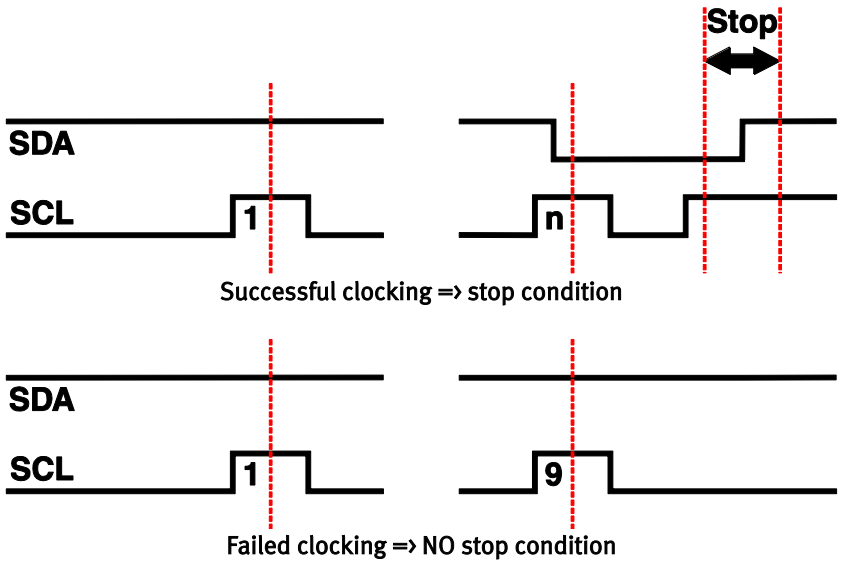
```
'' run sampling
Flag = XPort_4Samples ( 10, 02020202h, &        ' sample rate, shift down
S1$,S2$,S3$,S4$, &                              ' buffers
0,1,2,3, &                                      ' xport addresses
00001100b, 00001100b, 00001100b, 00001100b )    ' masks

'' stop sampling
Flag = XPort_4Samples ( 0 )

'' start new sampling
Flag = XPort_4Samples ( 10, 00000000h, &        ' sample rate, shift down
S1$,S2$,S3$,S4$, &                              ' buffers
0,1,2,3, &                                      ' xport addresses
00101001b, 10001100b, 00000110b, 01101100b )    ' masks
```

Updated functions

I2CL_CLK_RST

A = I2CL_CLK_RST ( )

Function: Tiger generates clocks until the slave pull down the SDA line.
This function is used as a bus master.



This function is used, if an I²C communication is interrupted and the slave is waiting to send the remainder of the data requested by the master. The bus must be recovered and the master has to restore control. The master sends some clocks and checks the SDA line. If the SDA line is pulled down, a stop condition is generated. The maximum clocks send by the master is 9. If there is no reaction after 9 clocks, no stop condition is generated. This process may need to be repeated some times.

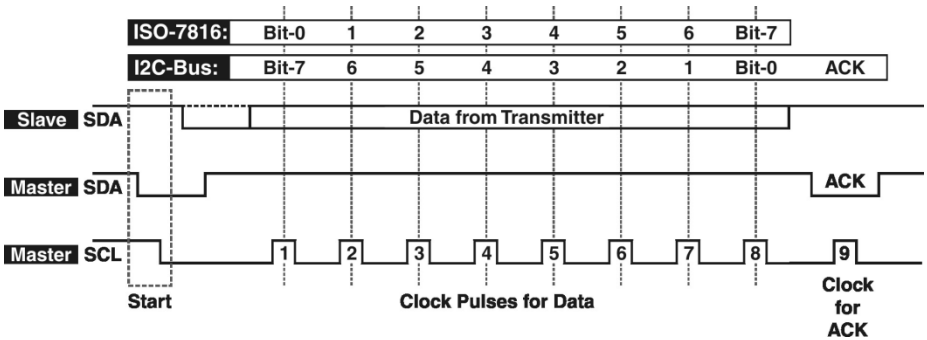
Parameters:

	B	W	L	S	F	Function value:
A	●	●	●	-	-	Clock number, when SDA line was pulled down from slave (1...9) or -1 if there was no reaction from slave device.

I2CL_Read$

```
A$      =   I2CL_Read$ ( nob )              ' READ Byte(s) from I2C bus
A$      =   I2CL_Read$ ( nob, ISO7816 )     ' READ Byte(s) from ISO 7816 bus

A$      =   I2CL_Read$ ( nob, ISO7816, cont )
```

Function: Reads the specified number of bytes from the I2C-Bus / ISO 7816 bus. This function is used as a bus master.



Read I2C bus / ISO 7816

Parameters:

	B	W	L	S	F	
nob	●	●	●	-	-	Number of bytes which are to be read: 0 ... 32
ISO7816	●	●	●	-	-	**7816**: ISO-7816 Mode **ELSE**: standard I2C Mode
cont	●	●	●	-	-	**0**: Standard mode. Last Byte is followed by ‹NAK› **ELSE**: Continuous mode. Last Byte is followed by ‹ACK›. Another I2CL_Read$ can be started. The last read process must be completed wit a ‹NAK›.
						Function value:
A$	-	-	-	●	-	Result string contains the received bytes.

Both bus lines SCL and SDA (CLOCK and DATA) are designed as “wired-and” signals. On every bus line there is:

- A pull-up resistor which pulls the line’s level to+5V as well as
- Open collector outputs of the bus devices
- High-resistive inputs of the bus devices

If all bus devices are inactive, i.e. no connected bus device affects a bus line, they are all applied to +5V level = “1”.

If one or several units with an open collector output pull a bus line to GND level (0V), all units see the bus signal “0” - “wired-and”. The level diagram for I2C bus / ISO 7816 reflects the circumstance as follows:

- Only 1 clock signal is depicted - the clock is always given by the master
- 2 data signals of each concerned bus device are depicted, so that the flow of information can be seen

Signal transmission on the I2C bus is not carried out at a fixed bits rate; however, it is limited for some chips and configurations. According to the “Speed”-parameter’s adjustment in I2CL_SETUP (...) I2CL_Read$ (...) accounts for the shortest bit time possible. Besides data transmission on the I2C bus is carried out completely statically - i.e. it can be stopped at any point and continued unaltered.

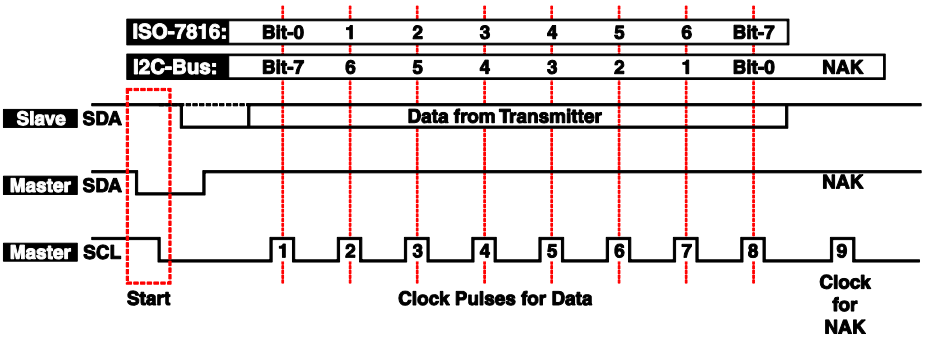
In the I2CL communication the Tiger determines the clock speed. There are situations where an I2C slave is not able to co-operate with the clock speed given by the Tiger and needs to slow down a bit. This is done by a mechanism referred to as **clock stretching**. An I2C slave is allowed to hold down the clock if it needs to reduce the bus speed. The Tiger on the other hand reads back the clock signal after releasing it to high state and waits until the line has actually gone high. The I2C specification does not specify any timeout conditions for clock stretching, i.e. any device can hold down SCL as long as it likes.

Note: The ISO-7816 transmission uses another RESET line for resetting the address counter. In case of an ISO 7816 application this line is implemented with an arbitrary Tiger pin or an extension pin and it is controlled by the BASIC program.

The “Answer-to-Reset” (ATR) procedure is standardized by ISO 7816-3.

There are several example programs with the prefix “I2CL_” and the extension “TIG”.

Continuous mode:



Last Byte in standard (non-continuous) mode

Note: In continuous mode, it is possible to read many Bytes from the I2C-Bus and execute more than one I2CL_Read$ without any start or stop condition. The ‹NAK› at the end of the I2CL_Read$ is disabled and a standard ‹ACK› is send. This means, more data will be received from the slave. There must follow one more I2CL_Read$. The last I2CL_Read$ command must be not in continuous mode.

Example for I2C_Read$ in continuous mode:

```
A$ = I2CL_READ$ (1,0,1)      ' READ 1 bytes from I2C-Bus in continuous
                             ' mode without <NAK>
A$ = I2CL_READ$ (1,0,0)      ' READ 1 bytes from I2C-Bus in standard mode
                             ' with a <NAK> at the end
```

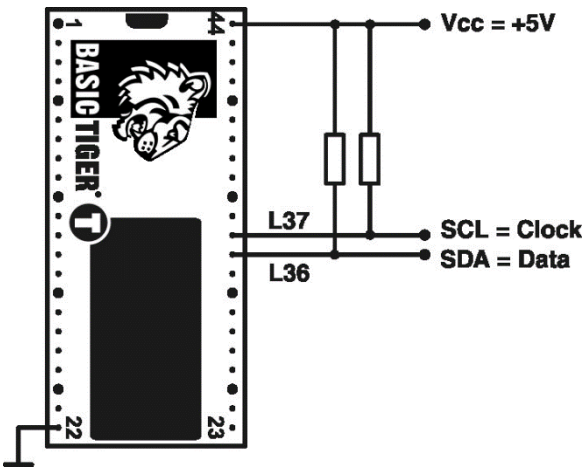
This is the same:

```
A$ = I2CL_READ$ (1,0,1)      ' READ 1 bytes from I2C-Bus in continuous
                             ' mode without <NAK>
A$ = I2CL_READ$ (1)          ' READ 1 bytes from I2C-Bus in standard mode
                             ' with a <NAK> at the end
```

I2CL_Setup

I2CL_Setup (Port, Clock_Pin, Data_Pin, Speed, RST_Pin, Clock_Bit_9_Delay)

Function: Specifies Tiger pins used and a possible timing speed reduction for low level functions I2C-Bus / ISO 7816 communication. All I2CL... functions are used as bus masters.



I2C bus / ISO 7816 bus (example)

Parameters:

	B	W	L	S	F	
Port	●	●	●	-	-	Internal port for signals: SDA and SCL
Clock_Pin	●	●	●	-	-	Clock output pin: (Bit-no.: 0...7) = clock generated by master
Data_Pin	●	●	●	-	-	DATA-I/O Pin: (Bit-no.: 0...7) = bidirectional
Speed	●	●	●	-	-	0 or › 254 = no speed reduction 1...254 = speed reduction
RST_Pin	●	●	●	-	-	Reset-pin for ISO 7816 (Bit-no.: 0...7, other: NO Reset Pin) (optional)
Clk_b_9_delay	●	●	●	-	-	speed reduction of ninth clock (ACK) (optional) 0 = no speed reduction 1...253 = speed reduction › 253 = same speed reduction like parameter 4

The Tiger supports I²C clock stretching from TAC/TA2 versions 1.16p or higher!

Example for I2CL_SETUP:

```
I2CL_SETUP ( 3, 7, 6, 0 )              'standard-setup

I2CL_SETUP ( 3, 7, 6, 0, 1 )           'fitting for ISO 7816 with
                                       'RST-Pin

I2CL_SETUP ( 3, 7, 6, 0, 1, 10 )       'with additional delay at the
                                       'ACK-CLK
```

This function line implements the following definition for the I2C bus low level and the ISO-7816 communication.

- Clock: port 3, bit 7
- Data: port 3, bit 6
- No speed reduction
- (RST: port 3, pin 1)
- (ACK-Bit-Delay: 10)

Every further I2C-Bus / ISO 7816 low level function access uses the definitions implemented by I2CL_Setup.

Both bus lines SCL and SDA (CLOCK and DATA) are designed as “wired-and” signals; on every bus line there are:

- A pull-up resistor which pulls the line’s level to +5V as well as
- Open collector outputs of the bus devices and
- High-resistive inputs of the bus devices

If all bus devices are inactive, i.e. all connected bus devices do not affect the bus line, every bus device is applied to a +5V level = “1”.

If one or several units pull a bus line with an open collector output to GND level (0V), all units see the bus signal “0” - “wired-and”. The level diagram for I2C bus / ISO 7816 mirrors this circumstance as follows:

- Only 1 clock signal is depicted - the clock is always given by the master
- 2 data signals of each concerned bus device are depicted, so that the flow of information can be watched

Signal transmission on the I2C bus is not carried out at a fixed bits rate; however, it is limited for some chips and configurations. According to the “Speed”-parameter’s adjustment in I2CL_SETUP (...), I2CL_Read$ (...) accounts for the shortest bit time possible. Besides data transmission on the I2C bus is carried out completely statically - i.e. it can be stopped at any point and continued unaltered.

In the I2CL communication the Tiger determines the clock speed. There are situations where an I2C slave is not able to co-operate with the clock speed given by the Tiger and needs to slow down a bit. This is done by a mechanism referred to as **clock stretching**. An I2C slave is allowed to hold down the clock if it needs to reduce the bus speed. The Tiger on the other hand reads back the clock signal after releasing it to high state and waits until the line has actually gone high. The I2C specification does not specify any timeout conditions for clock stretching, i.e. any device can hold down SCL as long as it likes.

The Clock-bit9-delay is a special parameter to set a longer period for the ninth clock (ACK). This is used, if the transferred data is checked or converted before the ACK is send. It may help avoiding timeouts. If it is not set (e.g. the parameter is not used in setup or it is set to a value above 253), the default speed reduction, which is set at parameter four in the setup, will be used.

Note: The ISO-7816 transmission uses another RESET line for resetting the address counter. In case of an ISO 7816 application this line is implemented with an arbitrary Tiger pin or an extension pin and it is controlled by the BASIC program.

The “Answer-to-Reset” (ATR) procedure is standardised by ISO 7816-3.

There are several example programs with the prefix “I2CL_” and the extension “TIG”.

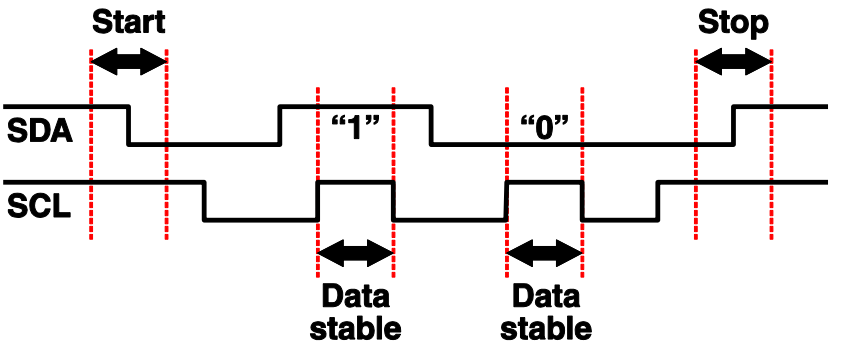
The parameter *Speed* slows down the bus speed and effects on:

- Data hold time
- Data set-up time
- Clock length
- Start / Stop condition (Delay between clock/data)

I2CL_Start

I2CL_Start (Dummy)

Function: Creates start condition on the I2C-Bus / ISO-7816 channel.
This function is used as a bus master.



I2C-Bus / ISO 7816 signals

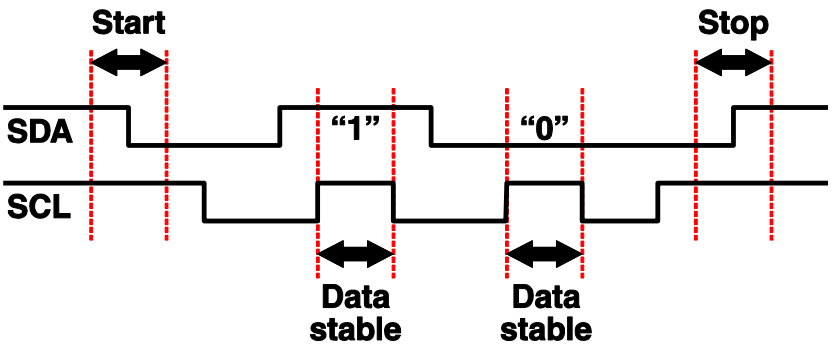
Parameters:

	B	W	L	S	F	
Dummy	●	●	●	-	-	Dummy value. Set to 0!
						Function value:
-	-	-	-	-	-	No function value

I2CL_Stop

I2CL_Stop (Dummy)
I2CL_Stop (Dummy, Clocks)

Function: Creates stop condition on the I2C-Bus / ISO-7816 channel.
This function is used as a bus master.



I2C-Bus / ISO 7816 signals

Parameters:

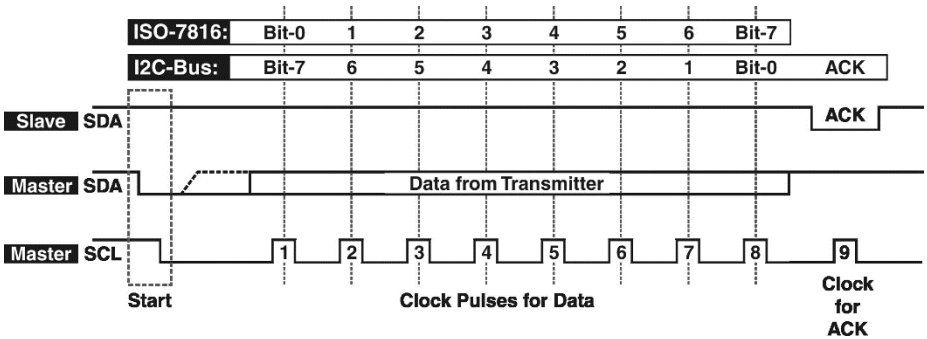
	B	W	L	S	F	
Dummy	●	●	●	-	-	Dummy value. Set to 0!
Clocks	●	●	●	-	-	Number of extra clocks
						Function value:
-	-	-	-	-	-	No function value

Clocks: In ISO-7816 mode it can be, that after finishing «Stop» procedure some more extra clocks should be sent without consideration of the SDA-line. One clock is approximately 20µs long.

I2CL_Write

```
NAK=    I2CL_Write ( A$ )                    ' Write A$ to I2C bus
NAK=    I2CL_Write ( N1, N2 )                ' Write N2 Bytes of N1 to I2C bus

NAK=    I2CL_Write ( A$, 7816 )              ' Write A$ to ISO 7816
NAK=    I2CL_Write ( N1, N2, 7816 )          ' Write N2 Bytes of N1 to ISO 7816

NAK=    I2CL_Write ( A$, 7816, Clocks )      ' Write A$ to ISO 7816
NAK=    I2CL_Write ( N1, N2, 7816, Clocks )  ' Write N2 Bytes of N1 to ISO 7816
```

Function: Writes the specified number of bytes to the I2C-Bus / ISO 7816 bus. This function is used as a bus master.



I2C bus / ISO 7816 write

Parameters:

	B	W	L	S	F	
A$	-	-	-	●	-	Transmission string: 0 ... 32 characters
N1	●	●	●	-	-	Numerical value, from which 1, 2, 3 or 4 bytes are sent
N2	●	●	●	-	-	Number of bytes which are to be transmitted: 1...4 bytes
7816	-	●	●	-	-	Identifier “ISO 7816” transmission format
Clocks	●	●	●	-	-	Number of extra clocks
						Function value:
NAK	●	●	●	-	-	Number of received ‹NAK› conditions during I2C bus transmission Of no relevance during ISO 7816 transmission

Clocks: In ISO-7816 mode it can be, that after finishing «Write» procedure some more extra clocks should be sent without consideration of the SDA-line. One clock is approximately 20µs long.

Both bus lines SCL and SDA (CLOCK and DATA) are designed as “wired-and” signals. On every bus line there is:

- A pull-up resistor which pulls the line’s level to+5V as well as
- Open collector outputs of the bus devices
- High-resistive inputs of the bus devices

If all bus devices are inactive, i.e. no connected bus device affects a bus line, they are all applied to +5V level = “1”.

If one or several units with an open collector output pull a bus line to GND level (0V), all units see the bus signal “0” - “wired-and”. The level diagram for I2C bus / ISO 7816 mirrors the circumstance as follows:

Only 1 clock signal is depicted - the clock is always given by the master
2 data signals of each concerned bus device are depicted, so that the flow of information can be seen

Signal transmission on the I2C bus is not carried out at a fixed bits rate; however, it is limited for some chips and configurations. According to the “Speed“-parameter’s adjustment in I2CL_SETUP (...), I2CL_Read$ (...) accounts for the shortest bit time possible. Besides data transmission on the I2C bus is carried out completely statically - i.e. it can be stopped at any point and continued unaltered.

Note: The ISO-7816 transmission uses another RESET line for resetting the address counter. In case of an ISO 7816 application this line is implemented with any Tiger pin or an extension pin and it is controlled by the BASIC program.

The “Answer-to-Reset” (ATR) procedure is standardised by ISO 7816-3.

There are several example programs with the prefix “I2CL_” and the extension “TIG”.

Also see: I2CL_START, I2CL_STOP, I2CL_READ, I2CL_WRITE, I2CL_RESULT

INPUT

INPUT #*Dev_No.* [, #*Sec_Adr*], *Variable*

INPUT is an input instruction that only works with a DEVICE driver.

The job of the device driver is to manage device-specific functions and possibly carry out device-specific operations independently. This greatly simplifies the programming of inputs and outputs. All types of devices, no matter how different they are, are addressed with the same BASIC program I/O instructions: PUT, GET, PRINT, INPUT, PRINT USING and INPUT LINE. Every I/O device is assigned its own device number in the program under which it is addressed.

Devices that have a number of channels, e.g. multi-channel A/D converters, multiple interfaces, machines with a number of controllable axes, etc., can also make use of secondary addresses with which the corresponding device channel is selected.

Depending on the device driver, it is possible to have channels which perform logic functions only and with no physical presence. For example, two channels can be used with the real time clock device driver: channel-0 = seconds, channel-1 = 1/100 seconds.

Finally, the device driver can be used to exchange data flows and control information. Data flows are the inputs and outputs that are normally sent to, or received from, a device in normal operation. Control information is used to adjust certain operating parameters or to inquire of certain device status.

Examples of device parameters are baud rate, voltage range, sample rate, operating mode etc. Examples of device status include buffer status, Ready/Busy, operating hours, error status, device driver version, etc.

No function code (function code = 0) is specified to use data traffic. There are a number of function codes to use control information, which depend on the device driver. Typical regularly used functions, such as buffer status, are controlled by uniform codes. The latest include file, "UFUNCn.INC", contains the available function code definitions. For more information, see ‘Device Driver’ in the Device Driver Manual.

INPUT takes data from the system user via an input device, which is assigned under **Dev_No**. The user's input data is not transferred to the **Variable** until there is a certain code. This limiter code can be that produced by pressing the comma key, a line return or a key returning a code from the value $00 to $1F. When this limiter code is entered, all of the data entered so far, is transferred.

Parameters:

	B	W	L	S	F	
#DEV_NO	●	●	●	-	-	specifies the number f the I/O device as defined at the start of the program with INSTALL_DEVICE
#Sec_Adr	●	●	●	-	-	Secondary address, specifies the channel number within this device driver with which communication is to take place. Without this optional information, the secondary address is assumed to be 0.
Variable	●	●	●	●	-	input buffer

The INPUT command waits until the user enters a limiter code, before completing. The following instruction in this task is only executed after the INPUT command has completed its function. All other tasks continue to run while the INPUT command is waiting for the limiter code, or until the timeout is reached. CPU resources are made available to other tasks.

The input length is limited to

a maximum of 255 characters
the maximum length of the variables
the size of the input buffer (depends on device driver).

Since TAC-version 1.16i you are able to set a timeout for INPUT via the SET_SYSVARN-function. If the timeout is set to zero, INPUT will wait infinitely. Else if the timeout is set to a valid value (1ms to 2.000.000.000ms), INPUT will wait for this time and in the case that no limiter code occurred, an empty string, or a zero (in case of a numerical variable) will be returned.

The syntax for setting the timeout is:

```
SET_SYSVARN ( INPUT_TIMEOUT, <TIMEOUT>)    ' <TIMEOUT> is the timeout in ms
```

Program example:

```
'------------------------------------------------------------------------
' Name: INPUT.TIG
'------------------------------------------------------------------------
#INCLUDE KEYB_PP.INC                    ' English keyboard layout
TASK MAIN                               ' begin task MAIN
  STRING A$                             ' var of type STRING
  LONG L                                ' var of type LONG
' install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' install LCD-driver (TINY-Tiger)
' INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  CALL INIT_KEYB (1)                    ' set keyboard etc.
  PRINT #1, "<1>input STRING"           ' output "input string"
  INPUT #1, A$                          ' STRING input (no echo!)
  PRINT #1, "Input = "; A$              ' output of A$
  PRINT #1, "input LONG-Value"          ' output "input value"
  INPUT #1, L                           ' LONG input (no echo!)
  PRINT #1, "Input ="; L                ' output of L

  SET_SYSVARN(INPUT_TIMEOUT, 1000)      ' set the timeout to one second
  INPUT #1, A$                          ' input with 1 second timeout
  PRINT #1, "Input ="; L                ' output of L
END
```

INPUT_LINE

INPUT_LINE #*Dev_No.* [, #*Sec_Adr*], *Variable*

INPUT_LINE is an input instruction that only works with a DEVICE driver.

The job of the device driver is to manage device-specific functions and possibly carry out device-specific operations independently. This greatly simplifies the programming of inputs and outputs. All types of devices, no matter how different they are, are addressed with the same BASIC program I/O instructions: PUT, GET, PRINT, INPUT, PRINT USING and INPUT LINE. Every I/O device is assigned its own device number in the program under which it is addressed.

Devices that have a number of channels, e.g. multi-channel A/D converters, multiple interfaces, machines with a number of controllable axes, etc., can also make use of secondary addresses with which the corresponding device channel is selected.

Depending on the device driver, it is possible to have channels which perform logic functions only and with no physical presence. For example, two channels can be used with the real time clock device driver: channel-0 = seconds, channel-1 = 1/100 seconds.

Finally, the device driver can be used to exchange data flows and control information. Data flows are the inputs and outputs that are normally sent to, or received from, a device in normal operation. Control information is used to adjust certain operating parameters or to inquire of certain device status.

Examples of device parameters are baud rate, voltage range, sample rate, operating mode etc. Examples of device status include buffer status, Ready/Busy, operating hours, error status, device driver version, etc.

No function code (function code = 0) is specified to use data traffic. There are a number of function codes to use control information, which depend on the device driver. Typical regularly used functions, such as buffer status, are controlled by uniform codes. The latest include file, "UFUNCn.INC", contains the available function code definitions. For more information, see ‘Device Driver’ in the Device Driver Manual.

The INPUT_LINE command reads a string of data from the device assigned to **Dev_No** into the value, **Variable**. The only delimiter code accepted by this command is the RETURN key code. Like the INPUT command, INPUT_LINE waits until input data is available, or the specified timeout is reached. The CPU time is placed at the disposal of other tasks during this period. Unlike the INPUT instruction, other break characters such as COMMA are also read in.

Parameters:

	B	W	L	S	F	
#DEV_NO	●	●	●	-	-	number f the I/O device as defined at the start of the program with INSTALL_DEVICE
#Sec_Adr	●	●	●	-	-	Secondary address, specifies the channel number within this device driver with which communication is to take place. Without this optional information, the secondary address is assumed to be 0.
Variable	-	-	-	●	-	stores input

The INPUT_LINE command waits until the user enters a limiter code, before completing. The following instruction in this task is only executed after the INPUT_LINE command has completed its function. All other tasks continue to run while the INPUT_LINE command is waiting for the limiter code. CPU resources are made available to other tasks.

The input length is limited by

a maximum of 255 characters
a maximum length of the variables
the size of the input buffer (depends on device driver).

Since TAC-version 1.16i you are able to set a timeout for INPUT_LINE via the SET_SYSVARN-function. If the timeout is set to zero, INPUT_LINE will wait infinitely. Else if the timeout is set to a valid value (1ms to 2.000.000.000ms), INPUT_LINE will wait for input for this time and in the case that no CR occurred, an empty string will be returned.

The syntax for setting the timeout is:

```
SET_SYSVARN( INPUT_TIMEOUT, <TIMEOUT>)      ' <TIMEOUT> is the timeout in ms
```

Program example:

```
'------------------------------------------------------------------------
' Name: INPUT_L.TIG
'------------------------------------------------------------------------
#INCLUDE KEYB_PP.INC                    ' English keyboard layout
TASK MAIN                               ' begin task MAIN
  STRING A$                             ' var of type STRING
' install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' install LCD-driver (TINY-Tiger)
' INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  CALL INIT_KEYB (1)                    ' set keyboard etc.
  PRINT #1, "<1>input STRING"           ' output to LC-display

  INPUT_LINE #1, A$                     ' line input (no echo!)
  PRINT #1, "Input = "; A$              ' output to LC-display

  SET_SYSVARN(INPUT_TIMEOUT,1000)       ' set the timeout to one second
  INPUT_LINE #1, A$                     ' line input (with 1 second timeout)
  PRINT #1, "Input = "; A$              ' output to LC-display
END                                     ' end task MAIN
```

INSTR

```
Pos = instr( Pattern$,     Source$,     PosStart, Option )
Pos = instr( PatternAddr,  Source$,     PosStart, Option, PatternLen )
Pos = instr( Pattern$,     SourceAddr , PosStart, Option, SourceLen )
Pos = instr( PatternAddr,  SourceAddr,  PosStart, Option, PatternLen, SourceLen )
```

Function: Searches for **Pattern$** starting from a **Position** in **Source$**, for a maximum length of 255 bytes. If the pattern or the source is stored in flash, the address and the length must be passed.
Returns the position at which **Pattern$** was found, or -1 if not found. The first position is 0.

Parameters:

	B	W	L	S	F	
Pattern$	-	-	-	●	-	pattern to be searched for
PatternAddr	-	-	●	-	-	flash address of pattern to be searched for
Source$	-	-	-	●	-	source to search in
SourceAddr	-	-	●	-	-	flash address of source to search in
PosStart	●	●	●	-	-	start position
Option	●	●	●	-	-	four option bits (see table below)
PatternLen	●	●	●	-	-	length of the pattern in flash
SourceLen	●	●	●	-	-	length of the source in flash
						Function value:
Pos	-	-	●	-	-	›=0: position of the first bit of the found pattern - -1: not found -2: invalid start position An automatical type conversion takes place during the assignment.

The option parameter has the following meaning:

Option binary / dec.	Search direction	7 or 8 bit mode	Take upper/lower case into account
0000 / 0	forwards	8-Bit	Yes
0001 / 1	backwards	8-Bit	Yes
0010 / 2	forwards	7-Bit	Yes
0011 / 3	backwards	7-Bit	Yes
0100 / 4	forwards	8-Bit	No
0101 / 5	backwards	8-Bit	No
0110 / 6	forwards	7-Bit	No
0111 / 7	backwards	7-Bit	No

INSTR searches for a pattern in a source over a length of 255 bytes. The search length of the function is limited to prevent time consuming searches, which cannot be interrupted. If the search is successful, the position in the source at which the first character of the pattern was found is returned. If the pattern is not found the result is -1. The assignment variable should thus be a (signed) LONG number.

Program example:

```
'------------------------------------------------------------------------
' INSTR.TIG
'------------------------------------------------------------------------
TASK MAIN                                ' begin task MAIN
  DATALABEL DL1,DL2,DL3

DL1::
  DATA BYTE "A T"
DL2::
  DATA BYTE "this is a test"
DL3::

' Install LCD driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' Install LCD driver (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  PRINT #1, INSTR("A T","this is a test",8,4)        ' output INSTR
  PRINT #1, INSTR(DL1, "this is a test",8,4,DL2-DL1) ' with flash-data
  PRINT #1, INSTR("A T", DL2,8,4,DL3-DL2)            ' with flash-data
  PRINT #1, INSTR(DL1, DL2,8,4,DL2-DL1,DL3-DL2)      ' with flash-data
END
```

For searching in a source of more than 255 bytes, call INSTR in a loop.

Program example:

```
for pos0 = 0 to len(source$)-len(pattern$) step 100h-len(pattern$)
  pos = instr( pattern$, source$, pos0, 0 )
  if 0 <= pos then
     pos0 = len(source$) ' exit loop
  endif
next
```

MODULO_INC

ValueNew = MODULO_INC (Value, Min, Max, Step)

Function: Increments or decrements the **Value** within the specified limits **Min** and **Max** by the value **Step**.

Parameters:

	B	W	L	S	F	
Value	●	●	●	-	-	current value to be in- or decreased
Min	●	●	●	-	-	lowest permitted value
Max	●	●	●	-	-	highest permitted value
Step	●	●	●	-	-	step for the increment/decrement. Negative values for decrement are only possible in LONG
						Function value:
ValueNew	●	●	●	-	-	New value after increment / decrement.

MODULO_INC changes the value by the specified step and prevents the value from getting outside the specified numerical range.

First Value is checked. If Value is higher than Max, it is set to Max. If it is lower than Min, it will be set to Min. Then Value is incremented or decremented by Step (Value + Step). In case Value exceeds Max during increment, (Max + 1 – Min) is subtracted from Value. If on decrement Value drops below Min, (Max + 1 – Min) is added to Value. The result is written to ValueNew.

Examples:

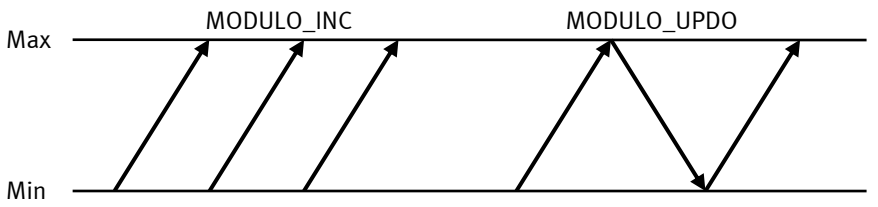
A =	Operation	B =
20	B = MODULO_INC (A, 12, 22, 1)	21
21	B = MODULO_INC (A, 12, 22, 1)	22
22	B = MODULO_INC (A, 12, 22, 1)	12

A =	Operation	B =
14	B = MODULO_INC (A, 12, 22, -1)	13
13	B = MODULO_INC (A, 12, 22, -1)	12
12	B = MODULO_INC (A, 12, 22, -1)	22

A =	Operation	B =
18	B = MODULO_INC (A, 12, 22, 2)	20
20	B = MODULO_INC (A, 12, 22, 2)	22
21	B = MODULO_INC (A, 12, 22, 2)	**12**
22	B = MODULO_INC (A, 12, 22, 2)	**13**

A =	Operation	B =
16	B = MODULO_INC (A, 12, 22, -2)	14
14	B = MODULO_INC (A, 12, 22, -2)	12
13	B = MODULO_INC (A, 12, 22, -2)	**22**
12	B = MODULO_INC (A, 12, 22, -2)	**21**

Difference in counting between MODULO_INC and MODULO_UPDO if the step is a positive value:



Program example:

```
'-----------------------------------------------------------------------
' Name: MOD_INC.TIG
'-----------------------------------------------------------------------
TASK MAIN                                ' begin task MAIN
  BYTE I, M
' install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' install LCD-driver (TINY-Tiger)
' INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  M = 18                                 ' begin with M=18
  PRINT #1, "begin 18, inc 1:"
  FOR I = 0 TO 5                         ' one line on LCD4x20
    M = MODULO_INC ( M, 12, 22, 1 )      ' increment
    PRINT #1, M;                         ' show M
  NEXT
  M = 16                                 ' begin with M=16
  PRINT #1, "<13><10>begin 16, inc 2:"
  FOR I = 0 TO 5                         ' enough for LCD4x20
    M = MODULO_INC ( M, 12, 22, 2 )      ' increment
    PRINT #1, M;                         ' show M
  NEXT
END                                      ' end task MAIN
```

Scan_or_Skip

```
New_Pos = Scan_or_Skip( SRC$, Charset$,  StartPos, Scan_Skip,[Reverse] )
New_Pos = Scan_or_Skip( SRC$, Flash,     StartPos, Scan_Skip,[Reverse] )
```

Function: Scans or skips given character collectives in strings. Scan_or_Skip is typically used to set up command interpreters and to analyse expressions and data structures.

Parameters:

	B	W	L	S	F	
SRC$	-	-	-	●	-	Source string
Flash	●	●	●	-	-	Flash-ADDR of Charset flag table
Charset$	-	-	-	●	-	Charset flag string , length: exactly 256 bytes There is a flag for every character code in the string: Flag value 0: Code does ***not*** belong to charset Flag value X: Code belongs to charset
StartPos	●	●	●	-	-	Start position in source string for scanning
Scan_Skip	●	●	●	-	-	Flag decides about way of scanning: Positive value -› SCAN Negative value -› SKIP
Reverse	●	-	-	-	-	Flag decides about the scan/skip direction: Zero: from “Pos” to the last char Non-Zero: reverse direction: from “Pos” to first char
						Function value:
NewPos	-	-	●	-	-	›=0: position of the first bit of the found pattern -1: not found -2: invalid start position

Scan_or_Skip checks a source string for specific character sets. During scanning the occurrence of characters of a certain character set is searched for starting from a given position. As soon as such a character is found scanning is completed; the character's position in the source string is transferred as a function value.

Example:

```
Blank_Flag$ = Fill$ ("00"%, 256)
Blank_Flag$ = NTOS$ (Blank_Flag$, 32, 1, 0FFH)
NPos = Scan_or_Skip ("The quick brown", Blank_Flag$, 0, 1) 'Scan for Blank
```

Blank_Flag$ is a flag string consisting of 256 bytes, all bytes = 00, apart from the byte at position 32 (20H). This is the flag for the ASCII space character. Therefore this flag string defines a character set which only consists of a space character.

After function call NPos has the value 3, which is the position of the character found first of the character set according to Blank_Flag$.

Now you could extend the character set with "separators" in general, e.g. all characters apart from letters and numbers.

```
Separator$ = "&
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' 00...0F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' 10...1F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' 20...2F
00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF  &      ' 30...3F
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  &      ' 40...4F
00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF  &      ' 50...5F
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  &      ' 60...6F
00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF  &      ' 70...7F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' 80...8F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' 90...9F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' A0...AF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' B0...BF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' C0...CF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' D0...DF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  &      ' E0...EF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"%       ' F0...FF
```

During "Skip" the occurrence of characters which are NOT part of a defined character set are scanned for starting from a given position. As soon as such a character is found, scanning is completed; the character's position in the source string is transferred as a function value.

Example:

```
Blank_Flag$ = Fill$ ("00"%, 256)
Blank_Flag$ = NTOS$ (Blank_Flag$, 32, 1, 0FFH)
NPos = Scan_or_skip ("  A   B  C    D", Blank_Flag$, 0, -1)     ' Scan for
                                                                ' NON Blank
```

After function call NPos has the value 2, which is the position of the character found first, which is NOT part of the character set according to Blank_Flag$.

Normally scanning/skipping starts at the given position and goes in direction of the end of the string. By setting the fifth parameter to another value than zero, the direction is reversed and scanning/skipping starts at the given position and walks to the beginning of the string.

SET_SYSVARN

FLG = SET_SYSVARN (*Function no, Parameter*)

Function: Sets the value of a LONG system variable. System variables of a numerical type (apart from REAL) will be set with this function.

Parameters:

	B	W	L	S	F	
Function_no	●	●	●	-	-	is a variable, a constant or an expression of the type BYTE, WORD, or LONG and is the number of the system variables
Parameter	●	●	●	-	-	can have different meanings or is sometimes a random number (Dummy).
Function value:						
Flg	-	-	●	-	-	is of the type LONG and returns the success status of the operation. Automatic type conversion during assignment. 0: o.k. 1: System variable does not exist. 2: Parameter wrong. This value is not possible.

Integrate the file 'DEFINE_A.INC' to use the function name since the function numbers can change during the further development of Tiger-BASIC. Functions of SET_SYSVARN:

Function name	No	2nd parameter	Description
SHIFT_SLOWER	‹00›	Value	Sets a parameter of the functions SHIFT_IN and SHIFT_OUT to slow down the clock-pulse of these functions. Range of values: 0...40

Function name	No	2nd parameter	Description
SHIFT_INTS	‹01›	Value	Disables internal Interrupts during SHIFT_OUT operation. 0: all enabled 1: 1ms Interrupts disabled (e.g. LCD) 2: same as “1” + Timer_A device drivers disabled 3: All Interrupts disabled
INPUT_TIMEOUT	‹02›	Value	Sets a timeout for the functions INPUT and INPUT_LINE Value = 0: no timeout is specified Range of values: 1...2.000.000.000

Tip

If you use the name, you do not need to adapt your program if changes are made in the numbers system. ‘THIS_TASK’ is used for the current task.

Program example:

```
'------------------------------------------------------------------------
' Name: SET_SYSVARN.TIG
' sets system variable for SHIFT_IN and SHIFT_OUT,
' to reduce clock speed
'------------------------------------------------------------------------
#INCLUDE DEFINE_A.INC

TASK MAIN                                ' Begin task MAIN
  LONG R                                 ' Result of SET_SYSVARN
  LONG D                                 ' var of type LONG
' Install LCD driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' Install LCD driver (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  DIR_PORT 8,0                           ' Port 8 is output
  LL_IPORT_OUT 8, 00000000b              ' idle state
  D = 0AAAAAAAAH                         ' set D
  SHIFT_OUT 8, 0, 1, D, 32               ' output of 32 bits
  R = SET_SYSVARN ( SHIFT_SLOWER, 15 )   ' set sytem paramerters
  PRINT #1, "result:";R
  SHIFT_OUT 8, 0, 1, D, 32               ' output slower
END                                      ' End task MAIN
```

SHIFT_OUT

SHIFT_OUT ***Log_iPortaddr, data pin, clock pin, variable, number***

Function: Clocked, serial output to external chips. SHIFT_OUT transfers the indicated **number** of bits of the **variable** through the **data pin**. For each bit of data, a clock impulse is produced by double inversion of the **clock pin**. Data pin and clock pin are on one internal port with the address **Log_iPortadr**.

Parameters:

	B	W	L	S	F	
Log_iPortaddr	●	●	●	-	-	Logical, internal port address
Data pin	●	●	●	-	-	Number of pin at which data bits are put out.
Clock pin	●	●	●	-	-	Number of pin that is used as clock pin.
Variable	●	●	●	●	●	Contains the data to be written at data pin.
Number	●	●	●	-	-	With numerical variables 'Number' determines how many bits are written. With a positive number, the least significant bit (LSB) will be sent first, with a negative number, the most significant bit (MSB) will be sent first. The number of bits to send is restricted to 32. Spare bits with BYTE or WORD are written as 0-bits. With variables of the type STRING, the given number is valid for every single byte and is restricted to 8. The whole string is always written. A REAL number (8 bit * 8 byte) can be directly shifted as a 64-bit value. Alternatively, a REAL number can be shifted, if it is first converted to a LONG number with the help of the functions RTL or LREAL and HREAL.

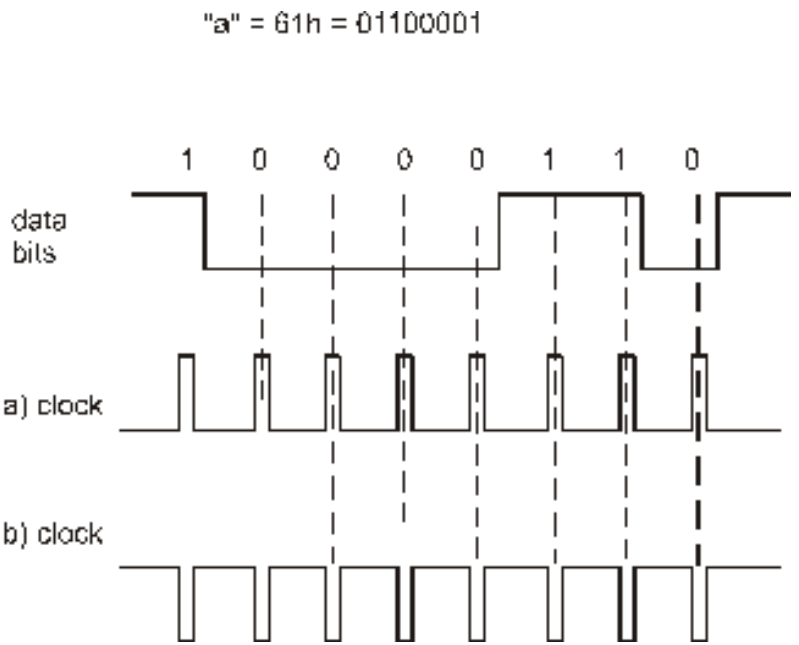
SHIFT_OUT writes a serial data stream clocked to an output-pin of an internal port. A second pin of the port is used as clock pin. The clock is set by the Tiger module. A clock impulse is generated by inverting the clock pin twice. The idle state

can therefore be preset. A SHIFT_OUT instruction writes a maximum of 32 bits with whole numbered numerical variables, always 64 bits with REAL numbers and with strings always the whole string. For strings, the quantity tells how many bits of each byte are being written.

The SHIFT_OUT instruction can be interrupted by system intern functions and device drivers to ensure real time operations or correct working of fast device drivers, e.g. TIMER_A device drivers. For the most devices this is no problem, because the timing is given by the clock. For real constant timings it is possible to disable the interrupts during the SHIFT_OUT operation with SET_SYSVARN. There are some devices, which have a timeout during the transmission phase. In this case, it could make sense to do this.

Applications of the SHIFT_OUT instruction are e.g. shift registers or connections of several modules, when the serial ports are already occupied for other purposes.

The picture shows the transfer of the character ‘a’ with high active (a) and with low active (b) level on the clock line.



On the Tiger 2® modules, the cycle time is 1.4 microseconds as a standard. With SET_SYSVARN, a system variable can be set that slows down the clock. Please take a look on the following tables:

SHIFT SLOWER	Frequency Tiger-1	Period Tiger-1	Clock length Tiger-1
0	217 KHz	4.6 µs	0.6 µs
1	128 KHz	7.8 µs	2.4 µs
2	98 KHz	10.2 µs	3.6 µs
5	54.9 KHz	18.2 µs	7.6 µs
10	32.2 KHz	31.1 µs	14.2 µs
15	22.62 KHz	44.2 µs	20.6 µs
20	17.54 KHz	57.0 µs	27.0 µs
25	14.25 KHz	70.2 µs	33.6 µs
30	12.05 KHz	83.0 µs	40.0 µs
35	10.40 KHz	96.2 µs	46.8 µs
40	9.19 KHz	108.8 µs	53.0 µs

SHIFT SLOWER	Frequency Tiger-2	Period Tiger-2	Clock length Tiger-2
0	690 KHz	1.45 µs	0.35 µs
1	590 KHz	1.7 µs	0.55 µs
2	500 KHz	2.0 µs	0.65 µs
5	312.5 KHz	3.2 µs	1.25 µs
10	192.3 KHz	5.2 µs	2.25 µs
15	138.5 KHz	7.2 µs	3.25 µs
20	108.7 KHz	9.2 µs	4.25 µs
25	89.3 KHz	11.2 µs	5.25 µs
30	75.8 KHz	13.2 µs	6.25 µs
35	65.7 KHz	15.2 µs	7.25 µs
40	58.1 KHz	17.2 µs	8.25 µs

SIGNEXT

SIGNEXT (Value, Number)

Function: Value extension with correct operational sign.
This function copies ***Number*** bits of a ***Value*** into a variable and fills excess digits with the most significant bit of the copied bits.

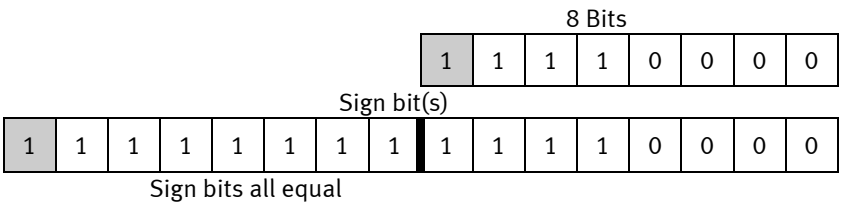
Parameters:

	B	W	L	S	F	
Value	●	●	●	-	-	is a variable, constant or expression of the type BYTE, WORD or LONG.
Number	●	●	●	-	-	is a variable, constant or expression of the type BYTE, WORD or LONG. This specifies the number of bits that the function
						Function value:
A	●	●	●	-	-	is of the type BYTE, WORD or LONG.

In the following examples, both a positive and a negative Value with different numbers of bits are signed in a BYTE:

BYTE	B before	After SIGNEXT (B, n)
SIGNEXT (B, 2)	00100101	00000001
SIGNEXT (B, 2)	00101111	11111111
SIGNEXT (B, 3)	00101001	00000001
SIGNEXT (B, 3)	00101101	11111101
SIGNEXT (B, 4)	10110001	00000001
SIGNEXT (B, 4)	10011001	11111001
SIGNEXT (B, 5)	11000111	00000111
SIGNEXT (B, 5)	11010111	11110111
SIGNEXT (B, 6)	10001111	00001111
SIGNEXT (B, 6)	11101111	11101111
SIGNEXT (B, 7)	10101101	00101101
SIGNEXT (B, 7)	01101101	11101101
BYTE after WORD		
SIGNEXT (L, 5)	11000111	00000000 00000111
SIGNEXT (L, 5)	11010111	11111111 11110111

W = SIGNEXT (B, 8)



The sign extension is used if signed values of different word lengths are to be processed in Tiger BASIC®, e.g. 5-bit measured values from a multimeter being used with 12-bit values of an A/D converter. Similarly, 4-Byte LONG values can be stored compactly in the Flash with the correct sign. Another application is the ability to serially transfer smaller values, by simply clipping the LONG number. Note that such values must have their sign restored before further processing.
There follows an example of what becomes of a negative LONG value in a WORD variable:

```
WORD W
LONG L
L = -87
W = L
PRINT #1, L, W ' L = -87, W = 65449
```

Program example:

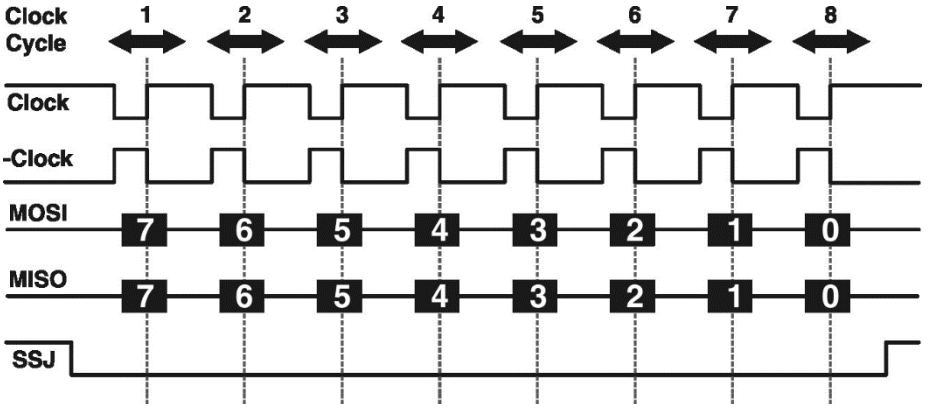
```
'--------------------------------------------------------------------
'Name: SIGNEXT.TIG
'--------------------------------------------------------------------
'This program shows examples for the function SIGNEXT(var,pos).
'The value of the bit 'pos' is set to every higher rated bit of
'the variable 'var'.
'--------------------------------------------------------------------
TASK Main                                'begin task MAIN
  BYTE B                                 'var of type BYTE
  WORD W                                 'var of type WORD
  LONG L                                 'var of type LONG
'install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
'install LCD-driver (TINY-Tiger)
'INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  USING "UH<8><8> 0.0.0.4 4"             'format string for output

  B = 70h                                'set B (112)
  PRINT USING #1, "<1>Bit: 7, B:";B      'output to LC-display
  B = SIGNEXT (B,7)                      'to BYTE: F0h (240)
  W = SIGNEXT (B,7)                      'to WORD: FFF0h (65520)
  L = SIGNEXT (B,7)                      'to LONG: FFFFFFF0h (-16)
  PRINT USING #1, "BYTE ="; B            'output to LC-display
  PRINT USING #1, "WORD ="; W            'output to LC-display
  PRINT USING #1, "LONG ="; L            'output to LC-display
  WAIT DURATION 3000                     'wait 3 sec

  W = 7000h 'set W (28672)
  PRINT USING #1, "<1>Bit:14, W:";W      'output to LC-display
  B = SIGNEXT (W,14)                     'to BYTE: 00h (0)
  W = SIGNEXT (W,14)                     'to WORD: F000h (61440)
  L = SIGNEXT (W,14)                     'to LONG: FFFFF000h (-4096)
  PRINT USING #1, "BYTE ="; B            'output to LC-display
  PRINT USING #1, "WORD ="; W            'output to LC-display
  PRINT USING #1, "LONG ="; L            'output to LC-display
  WAIT DURATION 3000                     'wait 3 sec

  L = 47654321h                          'set L (1197818657)
  PRINT USING #1, "<1>Bit:19, L:";L      'output to LC-display
  B = SIGNEXT (L,19)                     'to BYTE: 21h (33)
  W = SIGNEXT (L,19)                     'to WORD: 4321h (17185)
  L = SIGNEXT (L,19)                     'to LONG: FFFD4321h (-179423)
  PRINT USING #1, "BYTE ="; B            'output to LC-display
  PRINT USING #1, "WORD ="; W            'output to LC-display
  PRINT USING #1, "LONG ="; L            'output to LC-display
END                                      'end task MAIN
```

See also: SGN, INT, ABS

SPI_SETUP

```
FLG    =    SPI_SETUP  ( CLK_MOSI_Port, CLK_Pin, MOSI_Pin,        &
                         SSJ_Port, SSJ_Pin,                       &
                         MISO_Port, MISO_Pin, MSB_First)
```

Function: Specifies the SPI bus for the function: SPI_IO$ (...).



SPI Communication

Parameters:

	B	W	L	S	F	
CLK_MOSI_Port	●	●	●	-	-	Internal port for output signals: CLK + MOSI
CLK_Pin	●	●	●	-	-	Clock output pin: (Bit-No: 0...7) = clock generated by master
MOSI_Pin	●	●	●	-	-	DATA output pin: (Bit-No: 0...7) = master-OUT, slave-IN
SSJ_Port	●	●	●	-	-	Internal port for output signal: SSJ
SSJ_Pin	●	●	●	-	-	SSJ output pin: (Bit-No: 0...7) = Low-active CE for SPI device (-1: disable SSJ Pin)
MISO_Port	●	●	●	-	-	Internal port for input signal: MISO
MISO_Pin	●	●	●	-	-	DATA input pin: (Bit-No: 0...7)
MSB_First	●	●	●	-	-	Flag: 0=LSB first, X=MSB first
						Function value:
OK_FLAG	●	●	●	-	-	0 = everything OK, X = 1...n: nth parameter incorrect

The SPI_SETUP is executed once during program sequence and defines the I/O configuration for all following SPI accesses.

Example:

```
FLG  =  SPI_SETUP  ( 8,0,1, 8,3, 8,2, -1)
```

This function line implements the following definitions for SPI communication:

Clock	MOSI	MISO	SSJ
Port-8, Bit-0	Port-8, Bit-1	Port-8, Bit-2	Port-8, Bit-3

-1 ==› MSB first

```
FLG  =  SPI_SETUP  ( 8,0,1, 8,-1, 8,2, -1)
```

This function line implements the following definitions for SPI communication:

Clock	MOSI	MISO	SSJ
Port-8, Bit-0	Port-8, Bit-1	Port-8, Bit-2	no SSJ pin

-1 ==› MSB first

Also see: SPI_IO$

SYSVARN

RES = SYSVARN (FunctionNo, Parameter2)

Returns the Value of a LONG system variable. Numeric type system variables, other than real, are tested with this function. The test can also trigger system functions.

Parameters:

	B	W	L	S	F	
FunctionNo	●	●	●	-	-	is a variable, constant or expression of the type BYTE, WORD, LONG and is the number of the inquiry.
Parameter2	●	●	●	-	-	can have various meanings or is sometimes a random number (Dummy).
						Function value:
RES	-	-	●	-	-	is of the type LONG. An automatic type conversion takes place during the assignment.

The function numbers are assigned names in the Include file DEFINE_A.INC; these can be found in the table below.

Include the file 'DEFINE_A.INC' to use symbols, as function numbers may change in future developments of Tiger BASIC®. Functions of SYSVARN:

Symbol	No	2nd parameter	Description
RE_ID	‹00›	Taskname	Last runtime error code in the specified task
RE_CNT	‹01›	Taskname	Runtime error counter level
RE_SEMA	‹02›	Taskname	Current error flag (0=ok, 1=error)
RE_IDRES	‹15›	Taskname	Resets error flag and returns error code
CALL_LEVEL	‹16›	Taskname	Stack nesting level
DSTACK_FILL	‹17›	Taskname	Stack filling in bytes
DSTACK_FREE	‹18›	Taskname	Free stack space in bytes
DSTACK_SIZE	‹19›	Dummy	Stack size in bytes
DRAM	‹30›	0	Total DRAM in System
SRAM	‹31›	0	Total SRAM in System
FLASH_CHIPS	‹32›	Dummy	Number of Flash-Chips

Symbol	No	2nd parameter	Description
FLASH_SIZE	‹33›	Dummy	Size of Flash-Chips in bytes
FLASH_SEC	‹34›	Dummy	Number of sectors per chip
FLASH_SSIZE	‹35›	Dummy	Flash sector size
FLASH_ASEC	‹36›	Dummy	Number of Flash sectors
FLASH_GSIZE	‹37›	Dummy	Size of Flash memory in bytes
FLASH_DSEC	‹38›	Dummy	Number of Flash sectors for User-Data
FLASH_DSIZE	‹39›	Dummy	Size of Flash memory in bytes for User-Data
FLASH_DMODE	‹40›	Dummy	0=system waits during Flash operations 1=system continues to run during Flash operations
ACT_TASK	‹48›	Dummy	Current task number
TASK_PRIO	‹49›	Taskname	Current task priority of this task
SUM_PRIO	‹50›	Dummy	Sum total of all task priorities
NR_OF_TASKS	‹51›	Dummy	Number of tasks in program
NR_ACT_TASKS	‹52›	Dummy	Number of active tasks
BACKUP_RAM_SIZE	‹53›	Dummy	Size of Backup RAM memory in bytes (only available for Tiger plus modules)
START_INFO	‹65›	Dummy	Last reset cause: 0: PIN, power-on, software (Tiger 1 + 2) 1: Watchdog (Tiger 1 + 2) 26: PIN reset (Tiger plus) 27: Power-on reset (Tiger plus) 28: Software reset (Tiger plus) 29: Watchdog reset (Tiger plus) (requires Tiger plus 3.12b or higher)
TIGER_MODE	‹67›	Dummy	Tiger-Mode: 0: RUN-Mode 1: PC-Mode =Debug-MODE
TIGER_VERS	‹68›	Dummy	Tiger-Version
TIGER_MODULE	‹69›	Dummy	Tiger module Type: 003H = module family E3V 083H = module family TINY-Tiger 084H = module family TINY-Tiger 2 092H = module family ECONO-Tiger *plus*

Symbol	No	2nd parameter	Description
			093H = module family TINY-Tiger *plus* 094H = module family TINY-Tiger 2 *plus* 09AH = module family BASIC-Tiger *plus* 0AAH = module family A (BASIC-Tiger)
TAC_VERS	‹80›	Dummy	Version of the TAC files (numeric)
PC_MODE_PIN	‹81›	Dummy	State of PC-Mode-Pin 0: Low 1: High
BAS_PROG_LEN	‹83›	Dummy	Length of the BASIC program (without TAC files, device drivers and User Flash). *Not available with project model PM_SMALL_W*

Program example:

```
'-----------------------------------------------------------------------
' Name: SYSVARN.TIG
'-----------------------------------------------------------------------
#INCLUDE DEFINE_A.INC                    ' include global definitions

TASK MAIN                                ' begin task MAIN
' install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' install LCD-driver (TINY-Tiger)
' INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  CALL TEST                              ' call subroutine TEST
  PRINT #1, "<1>Stack-Level =";          ' output to LC-display
  PRINT #1, SYSVARN (CALL_LEVEL, -1)     ' get stack nesting level
  PRINT #1, "Stack-free  =";             ' output to LC-display
  PRINT #1, SYSVARN (DSTACK_FREE, -1)    ' get free stack space
END                                      ' end task MAIN

'-----------------------------------------------------------------------
' subroutine checks stack (recursive up to nesting level 10)
'-----------------------------------------------------------------------
SUB TEST                                 ' begin subroutine TEST
  PRINT #1, "<1>Stack-Level =";          ' output to LC-display
  PRINT #1, SYSVARN (CALL_LEVEL, -1)     ' get stack nesting level
  PRINT #1, "Stack-free  =";             ' output to LC-display
  PRINT #1, SYSVARN (DSTACK_FREE, -1)    ' get free stack space
  WAIT DURATION 2000                     ' wait 2 sec
  IF SYSVARN (CALL_LEVEL, -1) < 10 THEN  ' if nesting level < 10,
    CALL TEST                            ' call sub TEST recursive
  ENDIF                                  '
END                                      ' end subroutine TEST
```

Example to read out reset cause of Tiger module:

```
'--------------------------------------------------------------------------
'    Name:  Reset_cause.tig
'--------------------------------------------------------------------------
user_var_strict
#include define_a.inc

task main
  install_device #LCD, "lcd1.tdp"
  print #LCD, "<1>Reset cause:"

  switch sysvarn(START_INFO, 0)
  ' Tiger 1 + 2
  case 0: print #LCD, "reset"         ' PIN, power-on, software
  case 1: print #LCD, "watchdog"

  ' Tiger plus
  case 26: print #LCD, "PIN reset"
  case 27: print #LCD, "power-on reset"
  case 28: print #LCD, "software reset"
  case 29: print #LCD, "watchdog reset"

  ' Error cases
  case -1: print #LCD, "not supported<13><10>upgrade firmware"
  default: print #LCD, "unknown source"
  endswitch

  wait_duration 5000
  restart_prog()          ' force a software reset
end
```

SYSVAR$

RES = SYSVAR$ (*FunctionNo*, *Parameter2*)

Returns the **Number** Value of a STRING system variable.

Parameters:

	B	W	L	S	F	
FunctionNo	●	●	●	-	-	is a variable, constant or expression of the type BYTE, WORD, LONG and is the number of the inquiry.
Parameter2	●	●	●	-	-	can have various meanings or is sometimes a random number (Dummy).
						Function value:
RES	-	-	●	-	-	is of the type STRING

System variables of the type STRING are inquired with this function.

Include the file 'DEFINE_A.INC' to use symbols since the function numbers may change in further developments of Tiger BASIC®. Functions of SYSVAR$:

Symbol	No	2nd parameter	Description
RE_T_STRI	‹00›	Taskname	Error message text of last runtime error in task with number TaskNo
RE_C_STRI	‹01›	ErrCode	Error message text to error with No. ErrCode
RE_PNAME_STRI	‹83›	Dummy	Path and Name of Tiger Basic program

Program example:

```
'------------------------------------------------------------------------
' Name: SYSVAR$.TIG
'------------------------------------------------------------------------
TASK MAIN                                ' begin task MAIN
  FIFO BUFFER (32) OF BYTE               ' define FIFO-buffer
' install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' install LCD-driver (TINY-Tiger)
' INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  ON_ERRTASK_CALL Error                  ' call on error
  FOR I = 0 TO 35                        ' loop I from 0 to 35
    PRINT #1, "<1>I ="; I                ' output I to LC-display
    PUT_FIFO BUFFER, I                   ' create error (>31)
    WAIT DURATION 500                    ' wait 500 ms
  NEXT                                   ' next value for I
END                                      ' end task MAIN

'------------------------------------------------------------------------
' subroutine for error handling
'------------------------------------------------------------------------
SUB Error                                ' begin subroutine Error
  PRINT #1, "Error!"                     ' output "Error!"
  PRINT #1, SYSVAR$ (0,0)                ' output of error text
  WAIT DURATION 2000                     ' wait 2 sec
  ON_ERROR_RESET                         ' reset error flag
END                                      ' end subroutine Error
```

VAL_NUM

number = val_num (source$)
number = val_num (source$, offset, length)

Function: Returns the value of the first digit string from the character string. The returned value can be a BYTE, WORD or LONG number. VAL_NUM also recognizes hexadecimal numbers beginning with '$'.

Parameters:

	B	W	L	S	F	
Source$	-	-	-	●	-	source to find value in
offset	●	●	●	-	-	start position in source$ to search for digit string
length	●	●	●	-	-	number of bytes of source$ to search in
						Function value:
number	●	●	●	-	-	numerical value of first found digit string. value is zero in case of no digit string in source$

The string may contain characters before and after the number. If the string contains more than one number, the first is returned as the function value. The sign '$' initializes a hexadecimal number. An '$' sign with no following number is returned as value 0.

Tip

Characters which interfere when processing with VAL_NUM, can be removed with the function CONVERT$.

Examples:

```
X = val_num( "12" )                    ' Value is 12
X = val_num( "     12   13" )          ' Value is 12
X = val_num( "xyz 12 DM" )             ' Value is 12
X = val_num( "$0C" )                   ' Value is 12
X = val_num( "US$12" )                 ' Value is 12Hex = 18
X = val_num( "US$ 12" )                ' Value is 0
X = val_num( "012345", 3, 2 )          ' Value is 34
```

HDQ – 1-Wire

The HDQ and the 1-Wire interface both are interfaces using only 1 single pin. The HDQ interface has been developed by Texas Instruments and is used in many of their applications. Dallas developed the 1-Wire bus, which is slightly more complex because of its network addresses, but in return enables using several participants on the bus that can be individually addressed.

With the Tiger up to 8 of such interfaces can be driven simultaneously and parallel. So you can connect 8 different devices to 8 Tiger pins and change the functionality from HDQ to 1-Wire and vice versa even during run time.

Important: At the HDQ / 1-Wire pin a ***pull-up resistor*** to Vcc must be connected, otherwise the communication will not work. For example, for HDQ a 10k resistor and for 1-Wire a 4.7k resistor can be used. Both the GND lines certainly must be connected, too.

At the beginning a setup has to be executed for each channel, only then this pin can be used as interface. The following functions are available for communication:

HDQ_1WIRE_SETUP	=> defines port and pin of channel
HDQ_READ	=> reads byte through HDQ
HDQ_WRITE	=> writes byte through HDQ
ONEWIRE_RESET	=> issues reset signal through 1-Wire
ONEWIRE_READ	=> reads byte through 1-Wire
ONEWIRE_WRITE	=> writes byte through 1-Wire
ONEWIRE_READ_BIT	=> reads bit through 1-Wire (net addresses)
ONEWIRE_WRITE_BIT	=> writes bit through 1-Wire (net addresses)

Important: At 1-Wire, the basic communication is implemented. Realization of the communication protocol has to be written in BASIC. For the appropriate protocol please refer to the device's datasheet. At first the net address command has to be sent, which is followed by the function command with parameters etc. In BASIC you can read and write byte by byte through the bus. Should you intend to make use of the search algorithms for net addresses, additional functions for reading and writing bit by bit are available for you.

HDQ_ONEWIRE_SETUP

Flag = HDQ_ONEWIRE_SETUP (Port, Pin, Channel)

Function: One port and one pin for the HDQ and 1-Wire line is determined. Additionally a channel can be given, which enables to communicate on several channels and so the connect various devices.

Parameters:

	B	W	L	S	F	
Port	●	●	●	-	-	This port is used for the interface
Pin	●	●	●	-	-	This pin of above port is used for the interface
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
Flag	●	●	●	-	-	0 = Parameter OK -1 = Channel invalid -2 = Port invalid -3 = Pin invalid

! HDQ_1WIRE_SETUP must be executed at the beginning, before the interface can be used. There is no default pin for this interface.

If more than one interface is to be served, a channel should be set. If only one interface is used, setting the channel can be left out – it will be set to 0 by default.

! At the HDQ / 1-Wire pin a ***pull-up resistor*** to Vcc must be connected, otherwise the communication will not work. For example, for HDQ a 10k resistor and for 1-Wire a 4.7k resistor can be used. Both the GND lines certainly must be connected, too.

Example for setting up pin L80 (as channel 0):

```
'FLAG =  HDQ_1WIRE_SETUP(Port, Pin, Channel)
  res =  HDQ_1WIRE_SETUP(  8,   0)      ' End task MAIN
```

HDQ_READ

RES = HDQ_READ (Address, Channel)

Function: Reads one byte through the HDQ interface from *Address*. Devices that are accessed through HDQ have addresses that can be read out or written to, mostly registers or a RAM/ROM area.

Parameters:

	B	W	L	S	F	
Address	●	●	●	-	-	Address of connected device that is read out
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
RES	●	●	●	-	-	Read value stored in low byte of numerical variable: 0...255: Content of 'Address' -1 = Address too high -2 = Channel invalid Error codes only available with LONG variables!

! Before this function can be used, once the HDQ_1WIRE_SETUP has to be executed for the appropriate channel.

There is an R/W (Read or Write) bit in the HDQ protocol determining if data is to be written or read and that bit is added to the 7-bit address. This is covered by the function HDQ_READ, so please do not specify this bit yourself!

Program example:

```
task main
  byte addr
  long res
  string hdq1$(02Ch)
  hdq1$ = ""

  ' 1. without Channel ( Channel = 0 ) !!!

'FLAG =  HDQ_1WIRE_SETUP(Port, Pin, Channel)
  res =  HDQ_1WIRE_SETUP(  8,   0)

  if res >= 0 then
    for addr=0 to 02Ch
'     res = HDQ_READ(Address, Channel)
      res = HDQ_READ(addr)
      if res >= 0 then
        hdq1$ = hdq1$ + chr$(res)
      endif
    next
  endif



  ' 2. with Channel (Channel = 7)
'FLAG =  HDQ_1WIRE_SETUP(Port, Pin, Channel)
  res =  HDQ_1WIRE_SETUP( 8,    0,     7)

  if res >= 0 then
    for addr=0 to 02Ch
'     res = HDQ_READ(Address, Channel)
      res = HDQ_READ(   addr,    7)
      if res >= 0 then
        hdq1$ = hdq1$ + chr$(res)
      endif
    next
  endif


end
```

HDQ_WRITE

FLAG = HDQ_WRITE (Address, Data, Channel)

Function: A byte is written through the HDQ interface to *Address*. Devices that are accessed through HDQ have addresses that can be read out or written to, mostly registers or a RAM/ROM area.

Parameters:

	B	W	L	S	F	
Address	●	●	●	-	-	Address in connected device to which is written to (max. 7-bit address)
Data	●	●	●	-	-	Data byte that is written (max. 255)
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
FLAG	●	●	●	-	-	0 = OK -1= Address too high -2= Data byte too high -3= Channel invalid

! Before this function can be used, once the HDQ_1WIRE_SETUP has to be executed for the appropriate channel.

There is an R/W (Read or Write) bit in the HDQ protocol determining if data is to be written or read and that bit is added to the 7-bit address. This is covered by the function HDQ_READ, so please do not specify this bit yourself!

ONEWIRE_RESET

FLAG = ONEWIRE_RESET (Channel)

Function: Sends a 1-Wire signal. The bus is pulled to LOW for 480µs. A reset has to be sent before each (command) sequence. For the precise method of the 1-Wire protocol and the valid commands for the device, please refer to the appropriate datasheet.

Parameters:

	B	W	L	S	F	
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
FLAG	●	●	●	-	-	0= No device connected to bus ›0= At least 1 device connected to bus -1= Channel invalid

! Before this function can be used, once the HDQ_1WIRE_SETUP has to be executed for the appropriate channel.

ONEWIRE_READ

RES = ONEWIRE_READ (Channel)

Function: Reads 1 byte through the 1-Wire interface.

Parameters:

	B	W	L	S	F	
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
RES	●	●	●	-	-	Read value stored in low byte of numerical variable: 0...255: Content of 'Address' -1 = Channel invalid **ATTENTION:** 0FFh (-1) could be real data when using BYTE or WORD variable.

! Before this function can be used, once the HDQ_1WIRE_SETUP has to be executed for the appropriate channel.

Before a byte can be read, the slave has to know that he should transmit a byte, so the correct commands have to be written to the device first. These commands are e.g. 0CCh for skipping the net address (if only 1 slave is connected), then the command for reading (069h), followed by the address – now this address can be read out.

Program example:

```
#define NETADDRESS_SKIP 0CCh
#define IWIRE_FUNC_READ 69h
#define STARTADDR 0

task main
  byte x
  long res
  string onewire$(02Ch)
  onewire$ = ""

  ' 1. without Channel ( Channel = 0 ) !!!

'FLAG =  HDQ_1WIRE_SETUP(Port, Pin, Channel)
  res =  HDQ_1WIRE_SETUP(  8,   0)

  if res >= 0 then
'-------------------------------------------------------------------------
' 1. Send reset pulse
'-------------------------------------------------------------------------
'  FLAG = ONEWIRE_RESET(Channel)
    res = ONEWIRE_RESET()
    if res > 0 then
'-------------------------------------------------------------------------
' 2. Send net address command
'-------------------------------------------------------------------------
'    FLAG = ONEWIRE_WRITE(Value,          Channel)
      res = ONEWIRE_WRITE(NETADDRESS_SKIP)
      if res >= 0 then
'-------------------------------------------------------------------------
' 3. Function Command + Address
'-------------------------------------------------------------------------
'      FLAG = ONEWIRE_WRITE(Value,         Channel)
        res = ONEWIRE_WRITE(IWIRE_FUNC_READ)
        if res >= 0 then
'        FLAG = ONEWIRE_WRITE(Value,    Channel)
          res = ONEWIRE_WRITE(STARTADDR)
          for x = 0 to 1Ah
'-------------------------------------------------------------------------
' 4. Read out data
'-------------------------------------------------------------------------
'           RES = ONEWIRE_READ(Channel)
            res = ONEWIRE_READ()
            onewire$ = onewire$ + chr$(res)
          next
        endif
      endif
    endif
  endif


end
```

ONEWIRE_WRITE

FLAG = ONEWIRE_WRITE (Value, Channel)

Function: Writes 1 byte through the 1-Wire interface.

Parameters:

	B	W	L	S	F	
Value	●	●	●	-	-	Value to be sent through 1-Wire interface Value range: 0...255
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
FLAG	●	●	●	-	-	0: OK -1: Channel invalid -2: Value too high

! Before this function can be used, once the HDQ_1WIRE_SETUP has to be executed for the appropriate channel.

Writing does not refer only to writing to the chip, but is generally the sending of data. When an address on the chip is to be (re)written, special commands must be sent first – just like when reading. These commands are issued using the ONEWIRE_WRITE function.

Program example:

```
#define NETADDRESS_SKIP 0CCh
#define IWIRE_FUNC_READ 69h
#define STARTADDR 0

task main
  byte x
  long res
  string onewire$(02Ch)
  onewire$ = ""

  ' 1. without Channel ( Channel = 0 ) !!!

'FLAG =  HDQ_1WIRE_SETUP(Port, Pin, Channel)
  res =  HDQ_1WIRE_SETUP(  8,   0)

  if res >= 0 then
'-------------------------------------------------------------------------
' 1. Send reset pulse
'-------------------------------------------------------------------------
'  FLAG = ONEWIRE_RESET(Channel)
    res = ONEWIRE_RESET()
    if res > 0 then
'-------------------------------------------------------------------------
' 2. Send net address command
'-------------------------------------------------------------------------
'    FLAG = ONEWIRE_WRITE(Value,          Channel)
      res = ONEWIRE_WRITE(NETADDRESS_SKIP)
      if res >= 0 then
'-------------------------------------------------------------------------
' 3. Function Command + Address
'-------------------------------------------------------------------------
'      FLAG = ONEWIRE_WRITE(Value,          Channel)
        res = ONEWIRE_WRITE(IWIRE_FUNC_READ)
        if res >= 0 then
'        FLAG = ONEWIRE_WRITE(Value,    Channel)
          res = ONEWIRE_WRITE(STARTADDR)
          for x = 0 to 1Ah
'-------------------------------------------------------------------------
' 4. Daten auslesen
'-------------------------------------------------------------------------
'           RES = ONEWIRE_READ(Channel)
            res = ONEWIRE_READ()
            onewire$ = onewire$ + chr$(res)
          next
        endif
      endif
    endif
  endif


end
```

ONEWIRE_READ_BIT

RES = ONEWIRE_READ_BIT (Channel)

Function: Reads one single bit through the 1-Wire interface. Should the search algorithms for net addresses be used, reading and writing bit by bit are needed.

Parameters:

	B	W	L	S	F	
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
RES	●	●	●	-	-	Result (0 or 1) or -1: Channel invalid

! Before this function can be used, once the HDQ_1WIRE_SETUP has to be executed for the appropriate channel.

ONEWIRE_WRITE_BIT

FLAG = ONEWIRE_WRITE_BIT (Value, Channel)

Function: Writes one single bit through the 1-Wire interface. Should the search algorithms for net addresses be used, reading and writing bit by bit are needed.

Parameters:

	B	W	L	S	F	
Value	●	●	●	-	-	Value to be sent through 1-Wire interface Value range: 0 or 1
Channel	●	●	●	-	-	Channel 0...7 This parameter is *optional*; if it is left out it is set to 0 by default!
						Function value:
FLAG	●	●	●	-	-	0: OK -1: Channel invalid -2: Value invalid (too high)

! Before this function can be used, once the HDQ_1WIRE_SETUP has to be executed for the appropriate channel.

XPort 16 / 24

XSetup for 16-bit addresses

```
Flag      =   XSetup ( Bus_Port,     &
                       CTRL_Port,    &
                       Bit_ACLK,     &
                       Bit_DCLK,     &
                       Bit_INE,      &
                       CTRL2_Port,   &
                       Bit_Bus_CE,   &
                       CTRL3_Port,   &
                       Bit_ACLK2,    &
                       Save_Area)
```

Function: Defines bus and ctrl signal lines for the XPort system with **16-Bit addresses**.

Parameters:

	B	W	L	S	F	
Bus_Port	●	●	●	-	-	Port, which is used as an 8-bit DATA/ADDR bus: Port 6 or port 8
CTRL_Port	●	●	●	-	-	Port, which is used as a 3-bit CTRL bus for signals: ACLK, DCLK, -INE
Bit_ACLK	●	●	●	-	-	Bit-no: 0...7 for ACLK: ADDR clock
Bit_DCLK	●	●	●	-	-	Bit-no: 0...7 for DCLK: DATA clock
Bit_INE	●	●	●	-	-	Bit-no: 0...7 for -INE: Input-enable (low active)
CTRL2_Port	●	●	●	-	-	Port for CE signal which is used at XBus_OutR / XBus_InR. This signal is ***only*** needed for the functions XBus_OutR and XBus_InR. For applications which do not use these functions the signal should be set to a dummy value, i.e. a non-existent I/O pin, e.g. port 4, bit 7 (BASIC-Tiger, TINY-Tiger or Econo-Tiger).

	B	W	L	S	F	
Bit_Bus_CE	●	●	●	-	-	Bit-no: 0...7 for Bus_CE: BUS access CE signal. This signal is set to ACTIVE by the Tiger (i.e. its former level is INVERTED) during XBUS access. So the other unit knows that a BUS transmission takes place.
CTRL3_Port	●	●	●	-	-	Port for ACLK-2 signal which is used for 16- and 24-bit addresses.
Bit_ACLK2	●	●	●	-	-	Bit-no: 0...7 for ACLK-2: ADDR-2 clock. This address clock is used to generate bits 8...15 of the address.
SAVE_AREA	●	●	●	-	-	Start address of the save area. 256 XPorts are buffered in RAM. Only these addresses can be modified by the functions XSET, XRES and XINV. The least significant byte must be 0. Further signals, such as DATA direction can be defined and operated by the user in the BASIC program where applicable.
						Function value:
Flag	●	●	●	-	-	0 = OK, parameters accepted 1...5 = No. of the incorrect parameters

Note:

XSetup assigns Tiger I/O pins to the signals of the XPort system. The pins themselves are not altered, no direction assignment takes place and no value is set.

Therefore the common procedure is to:

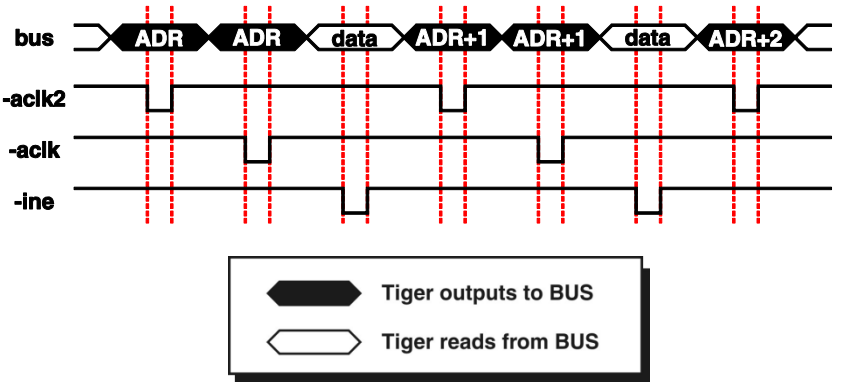
1.) XSetup	assign XPort pins
2.) Dir_Pin	set Ctrl pins to outputs
3.) Out	set pins to the defined level

Note:

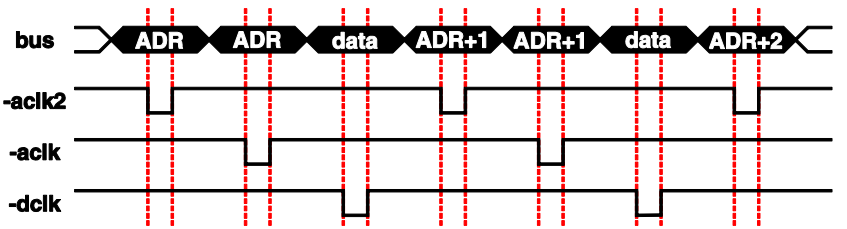
During an XBUS access, also other device drivers (LCD1, LCD2, parallel IN / OUT etc.) can use this bus. A running XBUS transmission is interrupted by such requests if necessary (XBUS_CE is set to

"inactive" during this time).Then the XBUS transmission is continued again.

The XPort system extends the Tiger I/O structure to up to 1,048,576 I/O lines. For this an 8-bit bus is used for transmitting address and data bytes as well as 4 ctrl lines for controlling the data stream (-ine, aclk-1, aclk-2, dclk). The I/O bus accesses for inputs and outputs to and from the XPort system:



Read access to XPorts in ascending ADDR order



Write access to XPorts in ascending ADDR order

The XPort system interacts directly with the I/O expansion modules of the EP line. Access to a Tiger system's XPort is carried out via only 1 BASIC line. Examples of circuitries and belonging program codes are presented in the description of the functions Xin/Xin$ and XOut.

To use the 16-bit addresses, an XSETUP is necessary to set the aclk-2 pin.

Example:

```
SUB initXPort
  byte blFlag

  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area

  wait duration 2000                ' wait until all modules are powered up
END
```

XSetup for 24-bit addresses

```
Flag      =   XSetup ( Bus_Port,     &
                       CTRL_Port,    &
                       Bit_ACLK,     &
                       Bit_DCLK,     &
                       Bit_INE,      &
                       CTRL2_Port,   &
                       Bit_Bus_CE,   &
                       CTRL3_Port,   &
                       Bit_ACLK2,    &
                       CTRL4_Port,   &
                       Bit_ACLK3,    &
                       Save_Area)
```

Function: Defines bus and ctrl signal lines for the XPort system with **24-bit addresses**.

Parameters:

	B	W	L	S	F	
Bus_Port	●	●	●	-	-	Port, which is used as an 8-bit DATA/ADDR bus: Port 6 or port 8
CTRL_Port	●	●	●	-	-	Port, which is used as a 3-bit CTRL bus for signals: ACLK, DCLK, -INE
Bit_ACLK	●	●	●	-	-	Bit-no: 0...7 for ACLK: ADDR clock
Bit_DCLK	●	●	●	-	-	Bit-no: 0...7 for DCLK: DATA clock
Bit_INE	●	●	●	-	-	Bit-no: 0...7 for -INE: Input-enable (low active)
CTRL2_Port	●	●	●	-	-	Port for CE signal which is used at XBus_OutR / XBus_InR. This signal is ***only*** needed for the functions XBus_OutR and XBus_InR. For applications which do not use these functions the signal should be set to a dummy value, i.e. a non-existent I/O pin, e.g. port 4, bit 7 (BASIC-Tiger, TINY-Tiger or Econo-Tiger).

	B	W	L	S	F	
Bit_Bus_CE	●	●	●	-	-	Bit-no: 0...7 for Bus_CE: BUS access CE signal. This signal is set to ACTIVE by the Tiger (i.e. its former level is INVERTED) during XBUS access. So the other unit knows that a BUS transmission takes place.
CTRL3_Port	●	●	●	-	-	Port for ACLK-2 signal which is used for 16- and 24-Bit addresses.
Bit_ACLK2	●	●	●	-	-	Bit-no: 0...7 for ACLK-2: ADDR-2 clock. This address clock is used to generate bits 8...15 of the address.
CTRL4_Port	●	●	●	-	-	Port for ACLK-3 signal which is used for 24-Bit addresses.
Bit_ACLK3	●	●	●	-	-	Bit-no: 0...7 for ACLK-3: ADDR-3 clock. This address clock is used to generate bit 16...23 of the address.
SAVE_AREA	●	●	●	-	-	Start address of the save area. 256 XPorts are buffered in RAM. Only these addresses can be modified by the functions XSET, XRES and XINV. The least significant byte must be 0. Further signals, such as DATA direction can be defined and operated by the user in the BASIC program where applicable.
						Function value:
Flag	●	●	●	-	-	0 = OK, parameters accepted 1...5 = No. of the incorrect parameters

Note:

XSetup assigns Tiger I/O pins to the signals of the XPort system. The pins themselves are not altered, no direction assignment takes place and no value is set.

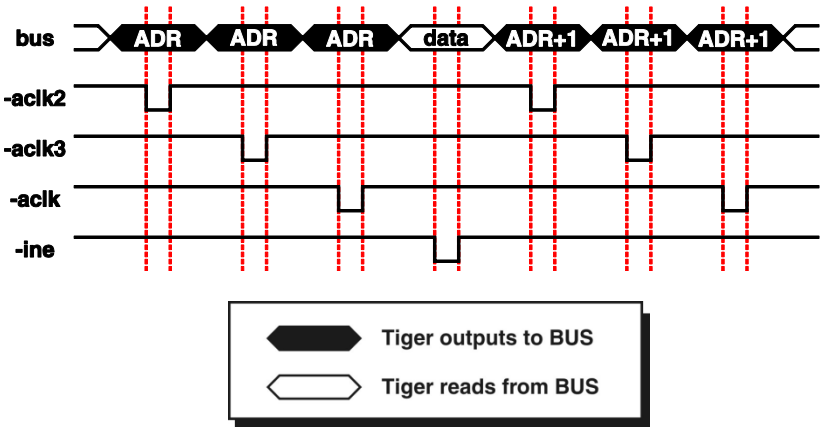
Therefore the common procedure is to:

1.) XSetup	assign XPort pins
2.) Dir_Pin	set Ctrl pins to outputs

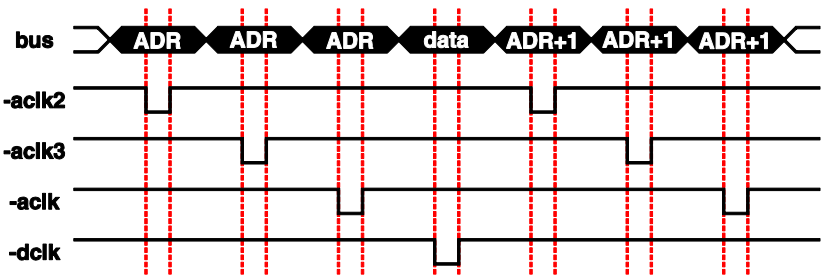
3.) Out set pins to the defined level

Note: During an XBUS access, also other device drivers (LCD1, LCD2, parallel IN / OUT etc.) can use this bus. A running XBUS transmission is interrupted by such requests if necessary (XBUS_CE is set to “inactive” during this time).Then the XBUS transmission is continued again.

The XPort system extends the Tiger I/O structure to up to 268,435,456 I/O lines. For this an 8-bit bus is used for transmitting address and data bytes as well as 5 ctrl lines for controlling the data stream (-ine, aclk-1, aclk-2, aclk-3, dclk). The I/O bus accesses for inputs and outputs to and from the XPort system:



Read access to XPorts in ascending ADDR order



Write access to XPorts in ascending ADDR order

The XPort system interacts directly with the I/O expansion modules of the EP line. Access to a Tiger system's XPort is carried out via only 1 BASIC line. Examples of circuitries and belonging program codes are presented in the description of the functions Xin/Xin$ and XOut.

To use the 24-bit addresses, an XSETUP is necessary to set the aclk-2 and aclk-3 pins.

Example:

```
SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Xin 1 byte

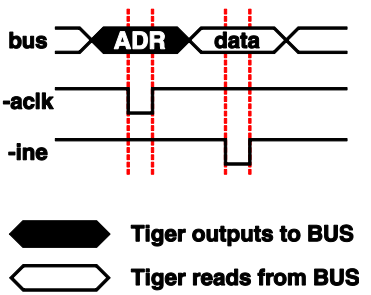
```
N = Xin (ADDR)                  ' <ADDR> <data>
N = Xin (ADDR, DELAY)           ' <ADDR> <DELAY> <data>
```

Function: Reads 1 byte from I/O expansion port (XPort) “ADDR” with an optional integrated delay. This slows down the bus transfer.

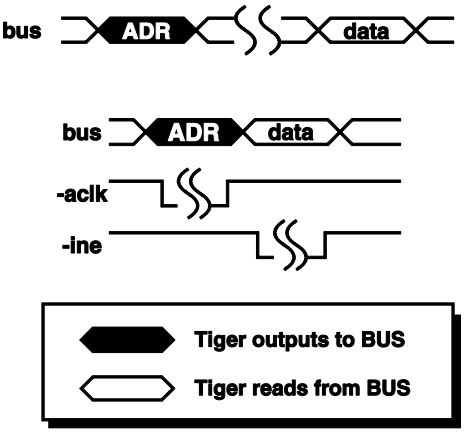
Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O Expansion port (XPort) address
DELAY	●	●	●	-	-	Delay to slow down the bus
						Function value:
N	●	●	●	-	-	1-byte result in numerical variable

Xin works with the XPort system’s default setting respectively with the settings explicitly made with XSETUP (...).



Xin (ADDR)



Xin (ADDR, DELAY)

The length of 1 DELAY is 150ns for Tiger-2. To generate a delay of 1,5µs, chose a DELAY value of 10. The Tiger-1 DELAY length is about 1,3µs. This function is typically used with Tiger-2, if devices running with Tiger-1 are too slow for the use with Tiger-2.

Example:

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte blInvalue

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  while 1=1
    blInvalue = XIN16(BASEADDRESS + TDR5000_OID_IN, 10)
  endwhile

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Xin16 1 byte

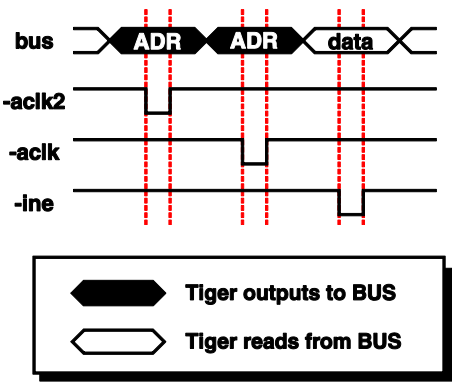
```
N = Xin16 ( ADDR )              ' <ADDR> <data> (16-bit address)
N = Xin16 ( ADDR, DELAY )       ' <ADDR> <DELAY> <data> (16-bit address)
```

Function: Reads 1 byte from I/O expansion port (XPort) “ADDR” with an optional integrated delay. This slows down the bus transfer.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O Expansion port (XPort) addr: 0 ... 65,535
DELAY	●	●	●	-	-	Delay to slow down the bus
						Function value:
N	●	●	●	-	-	1-byte result in numerical variable

Xin16 works with the settings explicitly made with XSETUP (...) for 16-bit addresses.



Xin16 (ADDR)

Example:

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte blInvalue

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  while 1=1
    blInvalue = XIN16(BASEADDRESS + TDR5000_OID_IN)
  endwhile

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Xin24 1 byte

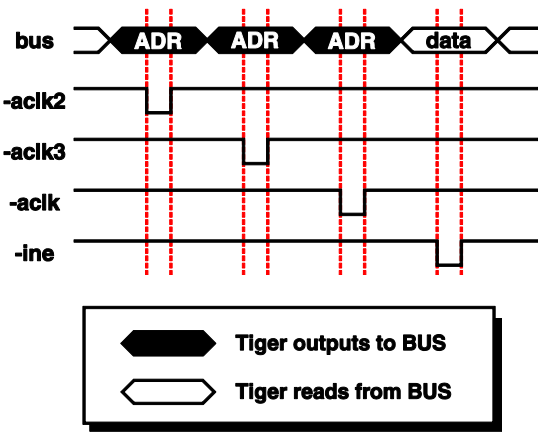
```
N = Xin24 ( ADDR )              ' <ADDR> <data> (24-bit address)
N = Xin24 ( ADDR, DELAY )       ' <ADDR> <DELAY> <data> (24-bit address)
```

Function: Reads 1 byte from I/O expansion port (XPort) “ADDR” with an optional integrated delay. This slows down the bus transfer.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O Expansion port (XPort) addr: 0 ... 16,777,215
DELAY	●	●	●	-	-	Delay to slow down the bus
						Function value:
N	●	●	●	-	-	1-byte result in numerical variable

Xin24 works with the settings explicitly made with XSETUP (...)for 24-bit addresses..



Xin24 (ADDR)

Example:

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte blInvalue

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  while 1=1
    blInvalue = XIN24(BASEADDRESS + TDR5000_OID_IN)
  endwhile

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Xin$ n bytes

```
X$ = Xin$ ( ADDR, NO, DELAY )   ' <ADDR> <data> <ADDR+1> <data> <ADDR+2> <data> ...
X$ = Xin$ ( -ADDR, NO, DELAY )      ' <ADDR> <data> <data> <data> ...
X$ = Xin$ ( 100h, NO, DELAY )       ' <data> <data> <data> ...
```

Function: Reads expansion port in different bus access modes with an optional integrated delay. This slows down the bus transfer.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O Expansion address: 0 ... 0FFh, respectively flag (>=100h)
NO	●	●	●	-	-	Number of bytes to read
DELAY	●	●	●	-	-	Optional: Delay to slow down the bus
						Function value:
X$	-	-	-	●	-	Result string with read data bytes

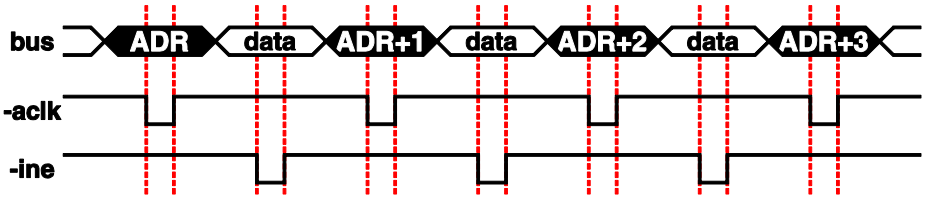
Xin$ works with the XPort system's settings explicitly made in XSETUP (...).

Xin$ is used for reading data from I/O expansion modules and other peripheral devices (memory, logic...). In its original form data is read from the ascending port addresses. This allows promptly scanning a number of input lines with one single BASIC instruction.

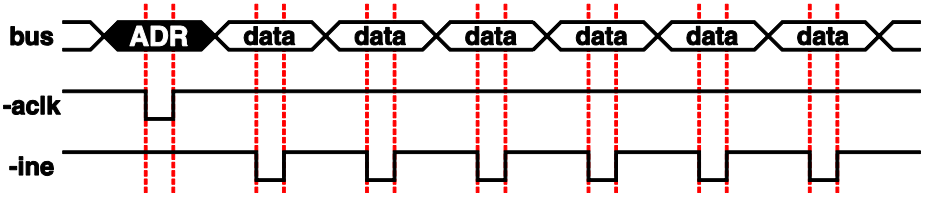
For fast parallel data transfer from peripheral units to the Tiger the following forms were added:

```
X$ = Xin$ (-ADDR, NO)           ' <ADDR> <data> <data> <data> ...
X$ = Xin$ (100h, NO)            ' <data> <data> <data> ...
```

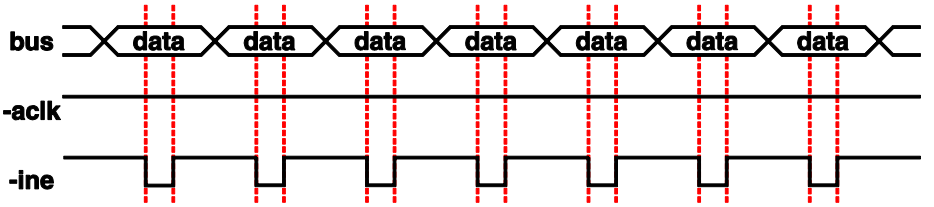
If needed, further control lines are controlled by the BASIC program, as is required for communicating with the peripheral unit (e.g. R/W or -CE signal).



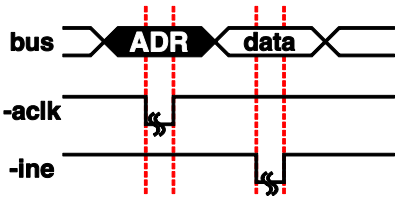
Xin$ (ADDR, NO)



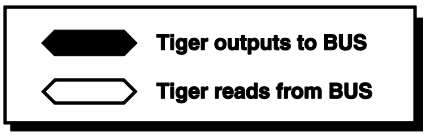
Xin$ (-ADDR, NO)



Xin$ (100h, NO)



Xin$ with Delay



The length of 1 DELAY is 150ns for Tiger-2. To generate a delay of 1,5µs, chose a DELAY value of 10. The Tiger-1 DELAY length is about 1,3µs. This function is typically used with Tiger-2, if devices running with Tiger-1 are too slow for the use with Tiger-2.

Example for XIN$ with Delay:

```
A$ = XIN$ ( 88h, 16, 14 )        ' reads in 16 Bytes from address 88H with a
                                 ' delay of 14 ~= 2µs (Tiger-2)
```

Example:

```
'-----------------------------------------------------------------------
'Name: XIN$.TIG
'direct read from phys. addresses of extended ports
'example uses the adresses of Plug & Play Lab's keyboard
'-----------------------------------------------------------------------
TASK MAIN                                'begin task MAIN
'install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
'install LCD-driver (TINY-Tiger)
'INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  LOOP 9999999                           'many loops
    A$ = XIN$ ( 88h, 16 )                'ext. inputs 88h..97h
    PRINT #1, "<1>";                     'erase LCD
    CALL SHOW_STRING_AS_HEX ( A$ )       'show string read
    WAIT_DURATION 200                    'wait 200 ms
  ENDLOOP
END                                      'end task MAIN


'-----------------------------------------------------------------------
'shows string as HEX bytes
'-----------------------------------------------------------------------
SUB SHOW_STRING_AS_HEX ( STRING S$ )
  BYTE I
  USING "UH<2><2>   0.0.0.0.2"           'format for HEX output
  FOR I = 0 TO LEN ( S$ ) - 1            'all bytes of the string
    PRINT_USING #1, ASC ( MID$ ( S$, I, 1 ) );'in HEX
  NEXT
  PRINT #1, ""                           'new line
END
```

Xin16$ n bytes

```
X$ = Xin16$ ( ADDR, NO, DELAY ) ' <ADDR> <data> <ADDR+1> <data> <ADDR+2> <data> ...
X$ = Xin16$ ( -ADDR, NO, DELAY )          ' <ADDR> <data> <data> <data> ...
X$ = Xin16$ ( 10000h, NO, DELAY )         ' <data> <data> <data> ...
```

Function: Reads expansion port in different bus access modes with an optional integrated delay. This slows down the bus transfer.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O Expansion address: 0 ... 0FFFFh, respectively flag (>=10000h)
NO	●	●	●	-	-	Number of bytes to read
DELAY	●	●	●	-	-	Optional: Delay to slow down the bus
						Function value:
X$	-	-	-	●	-	Result string with read data bytes

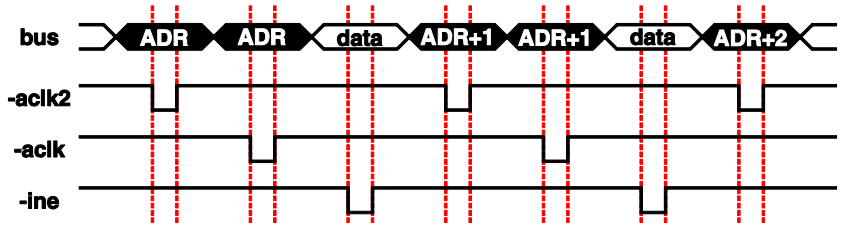
Xin16$ works with the XPort system's settings explicitly made in XSETUP (...) for 16-bit addresses.

Xin16$ is used for reading data from I/O expansion modules and other peripheral devices (memory, logic...). In its original form data is read from the ascending port addresses. This allows promptly scanning a number of input lines with one single BASIC instruction.

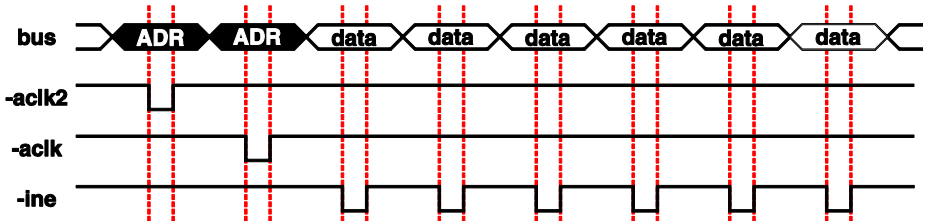
For fast parallel data transfer from peripheral units to the Tiger the following forms were added:

```
X$ = Xin16$ (-ADDR, NO)          ' <ADDR> <data> <data> <data> ...
X$ = Xin16$ (10000h, NO)         ' <data> <data> <data> ...
```

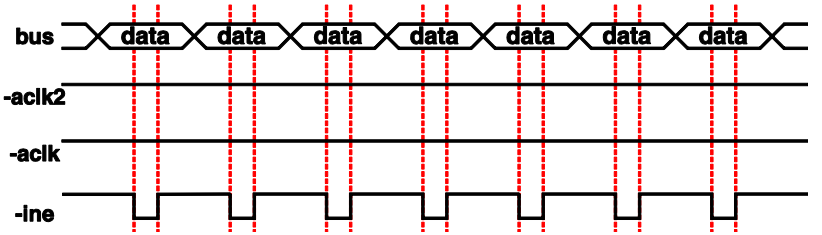
If needed, further control lines are controlled by the BASIC program, as is required for communicating with the peripheral unit (e.g. R/W or -CE signal).



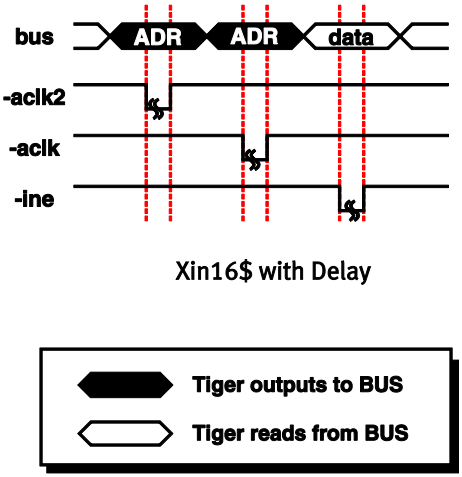
Xin16$ (ADDR, NO)



Xin16$ (-ADDR, NO)



Xin16$ (10000h, NO)



Xin16$ with Delay

The length of 1 DELAY is 150ns for Tiger-2. To generate a delay of 1,5µs, chose a DELAY value of 10. The Tiger-1 DELAY length is about 1,3µs. This function is typically used with Tiger-2, if devices running with Tiger-1 are too slow for the use with Tiger-2.

Example for XIN16$ with Delay:

```
A$ = XIN16$ ( 88h, 16, 14 )      ' reads in 16 Bytes from address 88H with a
                                 ' delay of 14 ~= 2µs (Tiger-2)
```

Example (subsequent addresses):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  string slIn$

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  while 1=1
    slIn$ = XIN16$(BASEADDRESS + TDR5000_OID_IN, 2)
  endwhile

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (one address):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

string slIn$(256)

TASK main
  byte control_A

  word llIbuFill

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  ' settings for serial interface SER-A on TDR5000
  control_A = 13h  ' 19200 Baud, 8 data bits, no handshake
  XOUT16(BASEADDRESS + TDR5000_SERA_CONTROL, control_A)


  while 1=1
    ' read no. of bytes in input buffer of SER A
    slIn$ = XIN16$(BASEADDRESS + TDR5000_SERA_FILL_LOW, 2)
    llIbuFill = nfroms (slIn$,0,2)
    if llIbuFill > 0 then
      slIn$ = XIN16$(0 - BASEADDRESS - TDR5000_SERA_DATA, llIbuFill)
    endif
  endwhile
end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Xin24$ n bytes

```
X$ = Xin24$ ( ADDR, NO, DELAY ) ' <ADDR> <data> <ADDR+1> <data> <ADDR+2> <data> ...
X$ = Xin24$ ( -ADDR, NO, DELAY )          ' <ADDR> <data> <data> <data> ...
X$ = Xin24$ ( 1000000h, NO, DELAY )       ' <data> <data> <data> ...
```

Function: Reads expansion port in different bus access modes with an optional integrated delay. This slows down the bus transfer.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O Expansion address: 0 ... 0FFFFFFh, respectively flag (>=1000000h)
NO	●	●	●	-	-	Number of bytes to read
DELAY	●	●	●	-	-	Optional: Delay to slow down the bus
						Function value:
X$	-	-	-	●	-	Result string with read data bytes

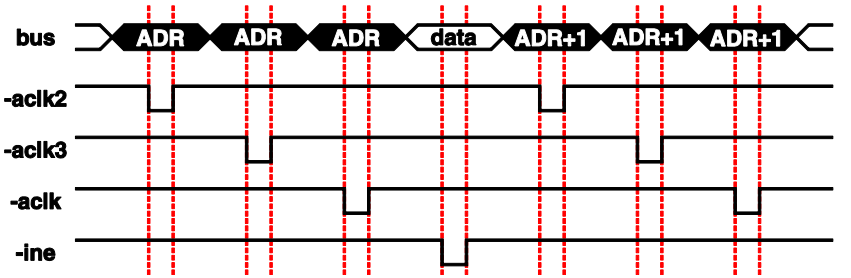
Xin24$ works with the XPort system's settings explicitly made in XSETUP (...) for 24-bit addresses.

Xin24$ is used for reading data from I/O expansion modules and other peripheral devices (memory, logic...). In its original form data is read from the ascending port addresses. This allows promptly scanning a number of input lines with one single BASIC instruction.

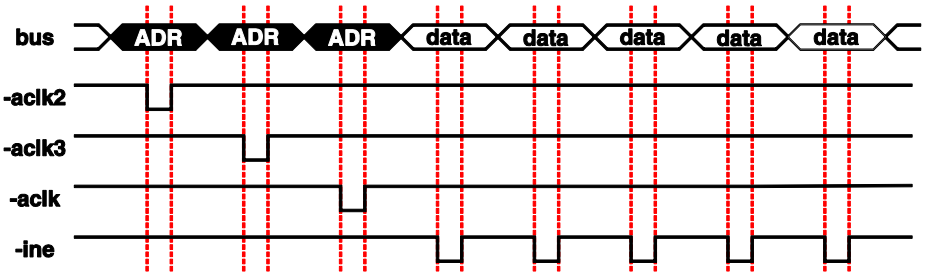
For fast parallel data transfer from peripheral units to the Tiger the following forms were added:

```
X$ = Xin24$ (-ADDR, NO)          ' <ADDR> <data> <data> <data> ...
X$ = Xin24$ (1000000h, NO)       ' <data> <data> <data> ...
```

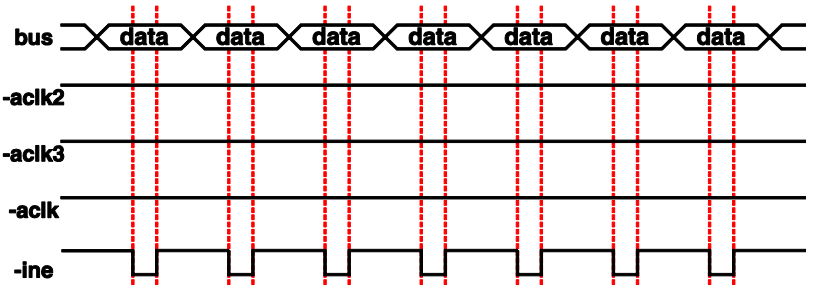
If needed, further control lines are controlled by the BASIC program, as is required for communicating with the peripheral unit (e.g. R/W or -CE signal).



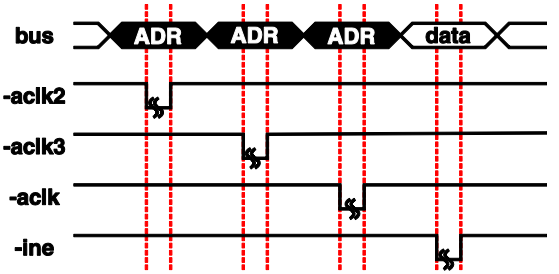
Xin24$ (ADDR, NO)



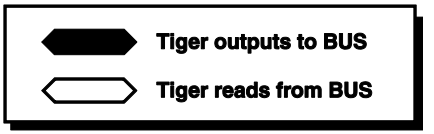
Xin24$ (-ADDR, NO)



Xin24$ (1000000h, NO)



Xin24$ with Delay



The length of 1 DELAY is 150ns for Tiger-2. To generate a delay of 1,5µs, chose a DELAY value of 10. The Tiger-1 DELAY length is about 1,3µs. This function is typically used with Tiger-2, if devices running with Tiger-1 are too slow for the use with Tiger-2.

Example for XIN24$ with Delay:

```
A$ = XIN24$ ( 88h, 16, 14 )      ' reads in 16 Bytes from address 88H with a
                                 ' delay of 14 ~= 2µs (Tiger-2)
```

Example (subsequent addresses):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  string slIn$

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  while 1=1
    slIn$ = XIN24$(BASEADDRESS + TDR5000_OID_IN, 2)
  endwhile

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (one address):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

string slIn$(256)

TASK main
  byte control_A
  word llIbuFill

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  ' settings for serial interface SER-A on TDR5000
  control_A = 13h  ' 19200 Baud, 8 data bits, no handshake
  XOUT24(BASEADDRESS + TDR5000_SERA_CONTROL, control_A)


  while 1=1
    ' read no. of bytes in input buffer of SER A
    slIn$ = XIN24$(BASEADDRESS + TDR5000_SERA_FILL_LOW, 2)
    llIbuFill = nfroms (slIn$,0,2)
    if llIbuFill > 0 then
      slIn$ = XIN24$(0 - BASEADDRESS - TDR5000_SERA_DATA, llIbuFill)
    endif
  endwhile
end

SUB initXPort
  byte blFlag
  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

XOut16 1 byte

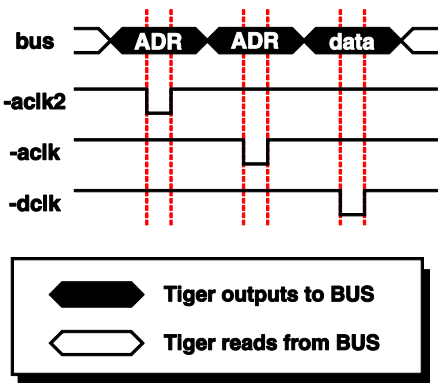
XOut16 (ADDR, N) **' ‹ADDR› ‹data› - write 1 Byte**

Function: Writes 1 byte to the I/O expansion port (XPort) “ADDR“.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O expansion port (XPort) addr: 0 ... 65,535
N	●	●	●	-	-	Low byte is written to XPort
						No function value

XOut16 works with the XPort system’s settings explicitly made in XSETUP (...) for 16-bit addresses.



XOut16 (ADDR, data)

Example:

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte channel
  byte blOutvalue

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  blOutvalue = 3
  while 1=1   ' for ever
    XOUT16(BASEADDRESS + TDR5000_OD_OUT_LOW, blOutvalue)
    XOUT16(BASEADDRESS + TDR5000_OD_OUT_HIGH, 00000000b)
    wait duration 100
    XOUT16(BASEADDRESS + TDR5000_OD_OUT_LOW, blOutvalue)
    XOUT16(BASEADDRESS + TDR5000_OD_OUT_HIGH, 00001111b)
    wait duration 100
    if blOutvalue >= 0c0h then
      blOutvalue = 3
    else
      blOutvalue = blOutvalue shl 1
    endif
  endwhile
end

SUB initXPort
  byte blFlag

  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area

  wait duration 2000                ' wait until all modules are powered up
END
```

XOut16 n bytes

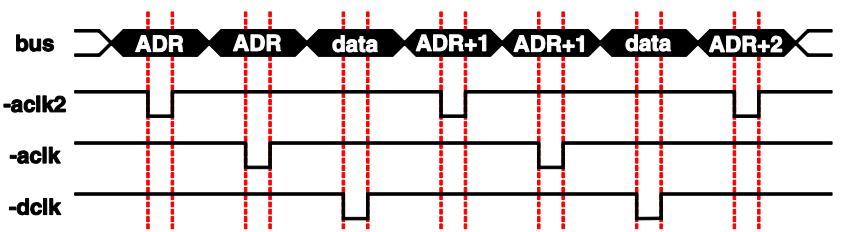
```
XOut16 ( ADDR, A$ )                 ' write many bytes to subsequent ADDRs
XOut16 ( ADDR, "" )                 ' write ONLY 1 ADDR cycle, NO data
XOut16 ( -ADDR, A$ )                ' write many bytes, 1 ADDR cycle only
XOut16 ( 1000000h, A$ )             ' write many bytes, NO ADDR cycle

XOut16 ( ADR, Flash, FLen )         ' write many bytes to subsequent ADDRs
XOut16 ( ADR, Flash, 0 )            ' write ONLY 1 ADDR cycle, NO data
XOut16 ( -ADR, Flash, FLen )        ' write many bytes, 1 ADDR cycle only
XOut16 (1000000h, Flash, FLen )     ' write many bytes, NO ADDR cycle
```

Function: Writes to I/O expansion ports (XPorts) in different modes.

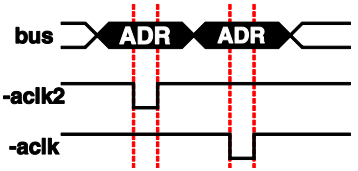
Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O expansion port (XPort) addr: 0 ... 0FFFFh
A$	-	-	-	●	-	Data string for sending to XPort(s)
Flash	●	●	●	-	-	Flash-ADDR at which data bytes to be sent reside
FLen	●	●	●	-	-	Number of bytes in Flash to be sent to XPort(s)
						No function value

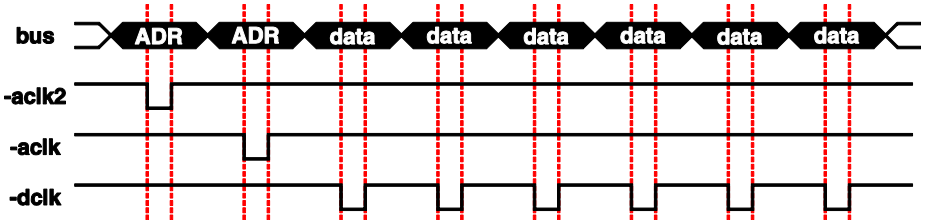
XOut16 works with the XPort system's settings explicitly made in XSETUP (...) for 16-bit addresses.



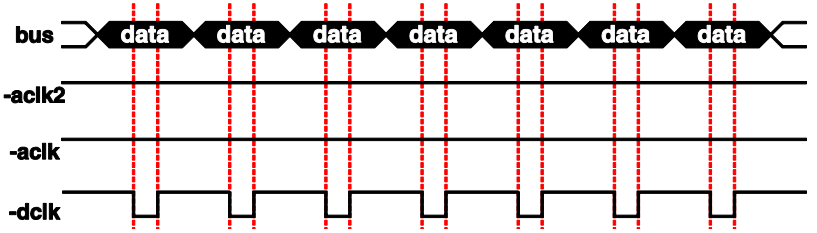
XOut16 (ADDR, A$) ● XOut16 (ADDR, Flash, FLen)



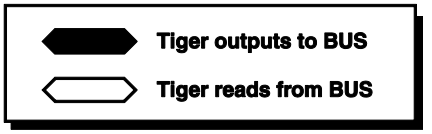
XOut16 (ADDR, "")



XOut16 (-ADDR, A$) ● XOut16 (-ADDR, Flash, FLen)



XOut16 (1000000h, A$) ● XOut16 (1000000h, Flash, FLen)



Example (subsequent addresses):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte channel
  byte blOutvalue
  string slXoutData$

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  blOutvalue = 3
  while 1=1   ' for ever
    slXoutData$ = chr$(blOutvalue) + chr$(00000000b)
    XOUT16(BASEADDRESS + TDR5000_OD_OUT_LOW, slXoutData$)
    wait duration 100

    slXoutData$ = chr$(blOutvalue) + chr$(00001111b)
    XOUT16(BASEADDRESS + TDR5000_OD_OUT_LOW, slXoutData$)
    wait duration 100

    if blOutvalue >= 0c0h then
      blOutvalue = 3
    else
      blOutvalue = blOutvalue shl 1
    endif
  endwhile
end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (one address):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte control_A
  byte blOutvalue
  string slXoutData$

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  ' settings for serial interface SER-A on TDR5000
  control_A = 13h  ' 19200 Baud, 8 data bits, no handshake
  XOUT16(BASEADDRESS + TDR5000_SERA_CONTROL, control_A)

  slXoutData$ = "Hello XOUT"
  XOUT16 (0 - BASEADDRESS - TDR5000_SERA_DATA, slXoutData$)

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (subsequent addresses from flash):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  string slXoutData$
  datalabel PATTERN1, PATTERN2

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  while 1=1   ' for ever
    XOUT16(BASEADDRESS + TDR5000_OD_OUT_LOW, PATTERN1, 2)
    wait duration 100

    XOUT16(BASEADDRESS + TDR5000_OD_OUT_LOW, PATTERN2, 2)
    wait duration 100
  endwhile

PATTERN1::
  data byte 055H
  data byte 005H
PATTERN2::
  data byte 0AAH
  data byte 00AH
end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (one address from flash):

```
user_var_strict
#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte control_A
  byte blOutvalue
  string slXoutData$
  datalabel FLASHDATA
  long llFlashAdr
  long llFlashLen

#ifdef TIGER_1
  llFlashAdr = FLASHDATA + 2
  peek_flash FLASHDATA, llFlashLen, 2
#endif
#ifdef TIGER_2
  llFlashAdr = FLASHDATA + 4
  peek_flash FLASHDATA, llFlashLen, 4
#endif

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  ' settings for serial interface SER-A on TDR5000
  control_A = 13h  ' 19200 Baud, 8 data bits, no handshake
  XOUT16(BASEADDRESS + TDR5000_SERA_CONTROL, control_A)

  XOUT16 (0 - BASEADDRESS - TDR5000_SERA_DATA, llFlashAdr, llFlashLen)
FLASHDATA::
  DATA STRING "Hello XOUT"
end

SUB initXPort
  byte blFlag

  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

XOut24 1 byte

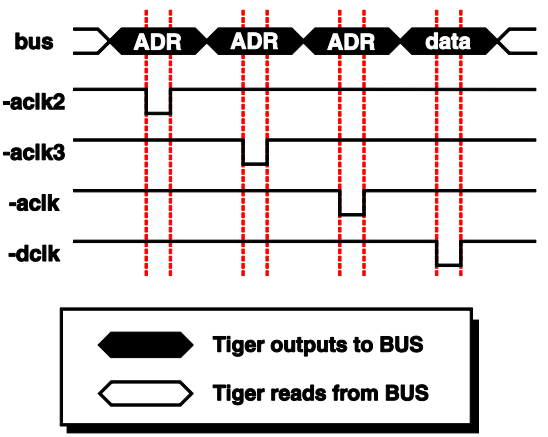
XOut24 (ADDR, N) **' ‹ADDR› ‹data› - write 1 Byte**

Function: Writes 1 byte to the I/O expansion port (XPort) “ADDR”.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O expansion port (XPort) addr: 0 ... 16,777,215
N	●	●	●	-	-	Low byte is written to XPort
						No function value

XOut24 works with the XPort system’s settings explicitly made in XSETUP (...) for 24-bit addresses.



XOut24 (ADDR, data)

Example:

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte channel
  byte blOutvalue

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  blOutvalue = 3
  while 1=1   ' for ever
    XOUT24(BASEADDRESS + TDR5000_OD_OUT_LOW, blOutvalue)
    XOUT24(BASEADDRESS + TDR5000_OD_OUT_HIGH, 00000000b)
    wait duration 100
    XOUT24(BASEADDRESS + TDR5000_OD_OUT_LOW, blOutvalue)
    XOUT24(BASEADDRESS + TDR5000_OD_OUT_HIGH, 00001111b)
    wait duration 100
    if blOutvalue >= 0c0h then
      blOutvalue = 3
    else
      blOutvalue = blOutvalue shl 1
    endif
  endwhile
end

SUB initXPort
  byte blFlag

  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

XOut24 n bytes

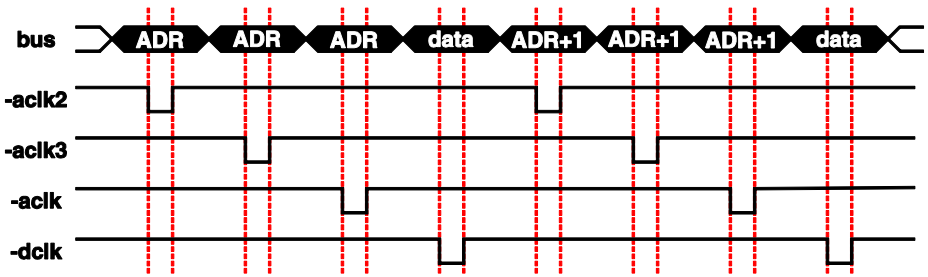
```
XOut24 ( ADDR, A$ )                 ' write many bytes to subsequent ADDRs
XOut24 ( ADDR, "" )                 ' write ONLY 1 ADDR cycle, NO data
XOut24 ( -ADDR, A$ )                ' write many bytes, 1 ADDR cycle only
XOut24 ( 1000000h, A$ )             ' write many bytes, NO ADDR cycle

XOut24 ( ADR, Flash, FLen )         ' write many bytes to subsequent ADDRs
XOut24 ( ADR, Flash, 0 )            ' write ONLY 1 ADDR cycle, NO data
XOut24 ( -ADR, Flash, FLen )        ' write many bytes, 1 ADDR cycle only
XOut24 (1000000h, Flash, FLen )     ' write many bytes, NO ADDR cycle
```

Function: Writes to I/O expansion ports (XPorts) in different modes.

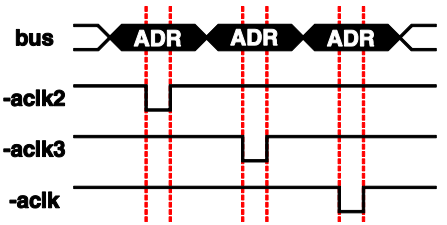
Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O expansion port (XPort) addr: 0 ... 0FFFFFFh
A$	-	-	-	●	-	Data string for sending to XPort(s)
Flash	●	●	●	-	-	Flash-ADDR at which data bytes to be sent reside
FLen	●	●	●	-	-	Number of bytes in Flash to be sent to XPort(s)
						No function value

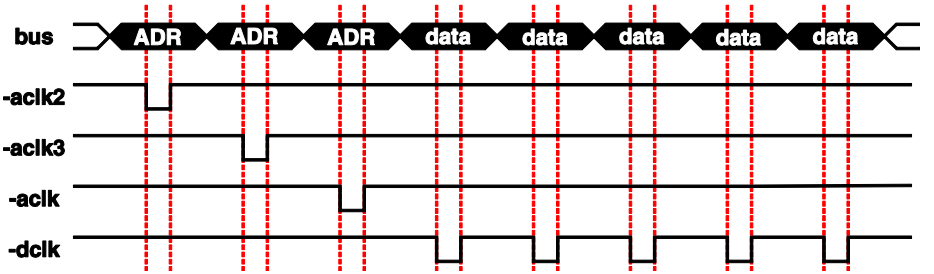
XOut24 works with the XPort system's settings explicitly made in XSETUP (...) for 24-bit addresses.



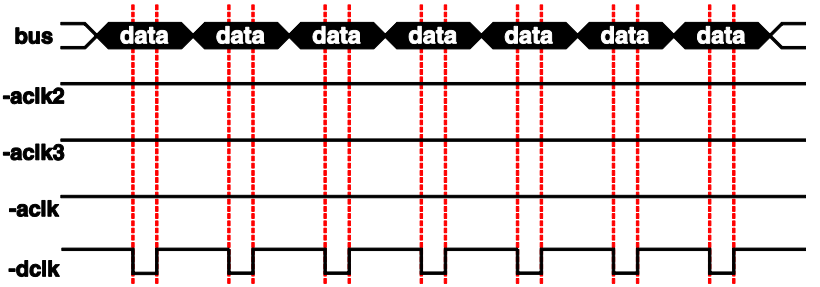
XOut24 (ADDR, A$) ● XOut24 (ADDR, Flash, FLen)



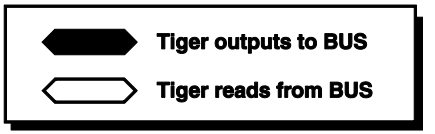
XOut24 (ADDR, "")



XOut24 (-ADDR, A$) ● XOut24 (-ADDR, Flash, FLen)



XOut24 (1000000h, A$) ● XOut24 (1000000h, Flash, FLen)



Example (subsequent addresses):

```
user_var_strict
#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte blOutvalue
  string slXoutData$
  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  blOutvalue = 3
  while 1=1   ' for ever
    slXoutData$ = chr$(blOutvalue) + chr$(00000000b)
    XOUT24(BASEADDRESS + TDR5000_OD_OUT_LOW, slXoutData$)
    wait duration 100

    slXoutData$ = chr$(blOutvalue) + chr$(00001111b)
    XOUT24(BASEADDRESS + TDR5000_OD_OUT_LOW, slXoutData$)
    wait duration 100

    if blOutvalue >= 0c0h then
      blOutvalue = 3
    else
      blOutvalue = blOutvalue shl 1
    endif
  endwhile
end

SUB initXPort
  byte blFlag

  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (one address):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte control_A
  byte blOutvalue
  string slXoutData$

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  ' settings for serial interface SER-A on TDR5000
  control_A = 13h  ' 19200 Baud, 8 data bits, no handshake
  XOUT24(BASEADDRESS + TDR5000_SERA_CONTROL, control_A)

  slXoutData$ = "Hello XOUT"
  XOUT24 (0 - BASEADDRESS - TDR5000_SERA_DATA, slXoutData$)

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (subsequent addresses from flash):

```
user_var_strict
#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  string slXoutData$
  datalabel PATTERN1, PATTERN2

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  while 1=1   ' for ever
    XOUT24(BASEADDRESS + TDR5000_OD_OUT_LOW, PATTERN1, 2)
    wait duration 100

    XOUT24(BASEADDRESS + TDR5000_OD_OUT_LOW, PATTERN2, 2)
    wait duration 100
  endwhile

PATTERN1::
  data byte 055H
  data byte 005H
PATTERN2::
  data byte 0AAH
  data byte 00AH
end

SUB initXPort
  byte blFlag

  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (one address from flash):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte control_A
  byte blOutvalue
  string slXoutData$
  datalabel FLASHDATA
  long llFlashAdr
  long llFlashLen

#ifdef TIGER_1
  llFlashAdr = FLASHDATA + 2
  peek_flash FLASHDATA, llFlashLen, 2
#endif

#ifdef TIGER_2
  llFlashAdr = FLASHDATA + 4
  peek_flash FLASHDATA, llFlashLen, 4
#endif

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  ' settings for serial interface SER-A on TDR5000
  control_A = 13h  ' 19200 Baud, 8 data bits, no handshake
  XOUT24(BASEADDRESS + TDR5000_SERA_CONTROL, control_A)

  XOUT24 (0 - BASEADDRESS - TDR5000_SERA_DATA, llFlashAdr, llFlashLen)


FLASHDATA::
  DATA STRING "Hello XOUT"
end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
```

```
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000               ' wait until all modules are powered up
END
```

XSet16, XSet24, XRes16, XRes24, Xinv16, Xinv24

```
XSet16 ( ADDR, Bit_Pos )      ' set 1 bit in XPort
XSet24 ( ADDR, Bit_Pos )      ' set 1 bit in XPort
XRes16 ( ADDR, Bit_Pos )      ' RESET 1 bit in XPort
XRes24 ( ADDR, Bit_Pos )      ' RESET 1 bit in XPort
Xinv16 ( ADDR, Bit_Pos )      ' Toggle 1 bit in XPort
Xinv24 ( ADDR, Bit_Pos )      ' Toggle 1 bit in XPort
```

Function: Manipulates a bit of an XPort output:

- Set bit
- Reset bit
- Invert bit

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O expansion port (XPort) addr
Bit_Pos	●	●	●	-	-	Bit-position: 0 ... 7
						No function value

There is an area of 256 addresses, which are buffered into the RAM of the Tiger. This area is set in the XSETUP. Bit manipulation with XSET, XRES and XINV are only allowed in this area. Manipulations on other addresses have no effect.

Program example:

```
TASK MAIN                       ' Beginning task MAIN
  XSET (2Bh, 2)                 ' Set Bit-2 in XPort 2Bh
  XRES (2Bh, 2)                 ' REST Bit-2 in XPort 2Bh
  XINV (2Bh, 2)                 ' Invert Bit-2 in XPort 2Bh
END                             ' Program end
```

Example (24 bit address):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 100h
#include TDR5000_D.INC

TASK main
  byte blOutvalue
  byte blPinNo

  blPinNo = 0

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module
  blOutvalue = 3
  XOUT24(BASEADDRESS + TDR5000_OD_OUT_LOW, "<0><0>")    ' init with 0


  while 1=1   ' for ever
    XRES24(BASEADDRESS + TDR5000_OD_OUT_LOW, blPinNo)
    blPinNo = modulo_inc(blPinNo, 0, 7, 1)
    XSET24(BASEADDRESS + TDR5000_OD_OUT_LOW, blPinNo)
    XINV24(BASEADDRESS + TDR5000_OD_OUT_HIGH, 0)
    wait duration 200
  endwhile
end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        100h)   ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Example (16 bit address):

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 100h
#include TDR5000_D.INC

TASK main
  byte blOutvalue
  byte blPinNo

  blPinNo = 0

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module
  blOutvalue = 3
  XOUT16(BASEADDRESS + TDR5000_OD_OUT_LOW, "<0><0>")    ' init with 0


  while 1=1   ' for ever
    XRES16(BASEADDRESS + TDR5000_OD_OUT_LOW, blPinNo)
    blPinNo = modulo_inc(blPinNo, 0, 7, 1)
    XSET16(BASEADDRESS + TDR5000_OD_OUT_LOW, blPinNo)
    XINV16(BASEADDRESS + TDR5000_OD_OUT_HIGH, 0)
    wait duration 200
  endwhile
end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        100h)   ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

XPin16

A = XPin16 (ADDR, Bit_Pos) ' reads 1 bit from XPort

Function: Reads a single bit of an XPort input.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O expansion port (XPort) address 0... 65535
Bit_Pos	●	●	●	-	-	Bit-position: 0 ... 7
						Function value:
A	●	●	●	-	-	Bit read from XPort, value: 0, 1

Example:

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte blInvalue

  CALL initXPort                ' initializes the X-Port
  XOUT16(BASEADDRESS, 1)        ' resets the module

  while 1=1
    blInvalue = XPIN16(BASEADDRESS + TDR5000_OID_IN, 0)
    blInvalue = XPIN16(BASEADDRESS + TDR5000_OID_IN, 1)
  endwhile

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

XPin24

A = XPIN24 (ADDR, Bit_Pos) ' reads 1 bit from XPort

Function: Reads a single bit of an XPort input.

Parameters:

	B	W	L	S	F	
ADDR	●	●	●	-	-	I/O expansion port (XPort) address 0... 16777215
Bit_Pos	●	●	●	-	-	Bit-position: 0 ... 7
						Function value:
A	●	●	●	-	-	Bit read from XPort, value: 0, 1

Example:

```
user_var_strict

#include define_a.inc
#define BASEADDRESS 0h
#include TDR5000_D.INC

TASK main
  byte blInvalue

  CALL initXPort                ' initializes the X-Port
  XOUT24(BASEADDRESS, 1)        ' resets the module

  while 1=1
    blInvalue = XPIN24(BASEADDRESS + TDR5000_OID_IN, 0)
    blInvalue = XPIN24(BASEADDRESS + TDR5000_OID_IN, 1)
  endwhile

end

SUB initXPort
  byte blFlag


  out 3, 00111000b, 00100000b       ' set port 3 bit 5 high, bit 4 and 3 low
  out 7, 10000000b, 0               ' set port 7 bit 7 low
  out 8, 10000000b, 0               ' set port 8 bit 7 low
  DIR_PIN 3, 3, 0                   ' set port 3 bit 3 as output (ACLK)
  DIR_PIN 3, 4, 0                   ' set port 3 bit 4 as output (DCLK)
  DIR_PIN 3, 5, 0                   ' set port 3 bit 5 as output (INE)
  DIR_PIN 7, 7, 0                   ' set port 7 bit 7 as output (ACLK3)
  DIR_PIN 8, 7, 0                   ' set port 8 bit 7 as output (ACLK2)

  blFlag = XSETUP ( 6,  & ' Bus Port
        3, &    ' CTRL-Port (for ACLK, DCLK, INE)
        3, &    ' Bit ACLK (address clock low address)
        4, &    ' Bit DCLK (data clock)
        5, &    ' Bit INE (input enable)
        4, &    ' CTRL2-Port (for CE)
        7, &    ' Bit Bus-CE (bus chip enable)
        8, &    ' CTRL3-Port (for ACLK2)
        7, &    ' Bit ACLK2 (address clock mid address)
        7, &    ' CTRL4-Port (for ACLK3)
        7, &    ' Bit ACLK3 (address clock high address)
        0)      ' save area
  wait duration 2000                ' wait until all modules are powered up
END
```

Overview of example programs

Name	Type	Description
XIN16_1BYTE.TIG	XIN16	Reads out 1 byte from 16-bit address X-Port
XIN16_1BYTE_DELAY.TIG	XIN16	Reads out 1 byte from 16-bit address X-Port with delay
XIN16_NBYTES_1.TIG	XIN16$	Reads many bytes from subsequent addresses
XIN16_NBYTES_2.TIG	XIN16$	Reads many bytes from ONLY 1 X-Port address
XIN16_NBYTES_1_DELAY.TIG	XIN16$	Reads many bytes from subsequent addresses with a Delay
XIN24_1Byte.TIG	XIN24	Reads out 1 byte from 24-bit address X-Port
XIN24_1BYTE_DELAY.TIG	XIN24	Reads out 1 byte from 24-bit address X-Port with delay
XIN24_NBYTES_1.TIG	XIN24$	Reads many bytes from subsequent addresses
XIN24_NBYTES_2.TIG	XIN24$	Reads many bytes from ONLY 1 X-Port address
XIN24_NBYTES_2_DELAY.TIG	XIN24$	Reads many bytes from ONLY 1 X-Port address with a Delay
XOUT16_1BYTE.TIG	XOUT16	Gives out 1 byte to 16-bit address X-Port
XOUT16_NBYTES_1.TIG	XOUT16	Writes many bytes to subsequent addresses from string
XOUT16_NBYTES_2.TIG	XOUT16	Writes many bytes to 1 address only from string
XOUT16_NBYTES_F1.TIG	XOUT16	Writes many bytes to subsequent addresses from flash
XOUT16_NBYTES_F2.TIG	XOUT16	Writes many bytes to 1 address only from flash
XOUT24_1Byte.TIG	XOUT24	Gives out 1 byte to 24-bit address X-Port
XOUT24_NBYTES_1.TIG	XOUT24	Writes many bytes to subsequent addresses from string
XOUT24_NBYTES_2.TIG	XOUT24	Writes many bytes to 1 address only from string
XOUT24_NBYTES_F1.TIG	XOUT24	Writes many bytes to subsequent addresses from flash
XOUT24_NBYTES_F2.TIG	XOUT24	Writes many bytes to 1 address only from flash
XSET16_XRES16_XINV16.TIG	XSET16 XRES16 XINV16	Bit manipulation on X-Ports (16-bit addresses)
XSET24_XRES24_XINV24.TIG	XSET24 XRES24 XINV24	Bit manipulation on X-Ports (24-bit addresses)
XPIN16.TIG	XPIN16	Reads a single but from XPort (16-bit addresses)

XPIN24.TIG	XPIN24	Reads a single but from XPort (24 bit addresses)
XIN$_DELAY.TIG	XIN$	Reads many bytes from subsequent addresses with a Delay

Documentation History

Version of Documentation	Description / Changes
001	First version
002	Uninstall_device
003	INPUT / INPUT_LINE timeout SHIFT slower (SET_SYSVARN)
005	SCAN_OR_SKIP INSTR
006	XPORT_4SAMPLES VAL_NUM
007	I2CL Clock Stretching
008	FPGA_SETUP & FPGA_PROGRAM Recursive Task switching: ENABLE_TSW_NESTED, DISABLE_TSW_NESTED, RESET_TSW_NESTED
009	XIN$, XIN16$ and XIN24$ with Delay
010	RELEASE_TSW_NESTED, READ_TSW_NESTED
011	SYSVAR$: RE_PNAME_STRI
012	SHIFT_OUT Timing Tiger-1 and reference to disable interrupts during SHIFT_OUT instruction SET_SYSVARN: SHIFT_INTS added.
013	New function UPDATE_ME_FILE_INF$
014	Updated function MODULO_INC
015	Included up-to-date list of preprocessor instructions
016	New function: PROGHASH SYSVARN new function parameters Project model: PM_SMALL_W2
017	Telephone number changed
018	I2CL Speed reduction
019	I2CL RST_Pin

020	SCALE SYSVARN Module Type I2CL_SETUP I2CL_WRITE I2CL_STOP SHIFT_OUT SIGNEXT SPI_SETUP SYSVAR$
021	Added BACKUP_RAM_SIZE in SYSVARN
022	Added START_INFO in SYSVARN
023	SYSVARN: RE_TAC_VERS renamed to TAC_VERS Functions sorted alphabetically VCDIFF_ENCODE$ + VCDIFF_DECODE$