

SER2 serial interfaces through software

Blank Page

Index

Index	3
SER2 - Serial interfaces through software	4
User-Function-Codes of the SER2_pp_xx.TDD	9
Documentation History	14

SER2 - Serial interfaces through software

This device driver enables an asynchronous serial input and output on internal I/O-pins. The interface has been implemented purely in the software. When installing the driver the file name determines at which pins the serial input and output takes place. The baud rate is determined by the TIMERA setting as well as the prescaler of the device driver.

The driver can be set to individual requirements:

- RxD + TxD: activate/deactivate individual channels.
- RxD + TxD: each with 256 byte FiFo buffer.
- RxD + TxD: with flow control: RTS / CTS activate/deactivate.
- TxD: RS-485 bus access control TE activate/deactivate.
- Data-Bits: Data format: 1...8 Bits.
- Parity-Bit: No, Even, Odd, Mark, Space.
- Baud rates: quasi-infinitely variable baud rates via TIMERA and Prescaler in the range from 12 kBd -send (with 1 x TxD) up to 3 Bd.
- Level: TRUE + INVERSE level possible for RS-232 with/without power driver.
- PINs: RxD, TxD, RTS, CTS and TE can be laid to almost any I/O-pin of the Tiger.
- Channels: up to 8 serial channels (RxD / TxD in random mixture).

Note: SER2_XX.TDD puts much more strain on the CPU than a driver such as SER1B_xx.TDD since several System-Task calls are carried out for every single bit. The following should therefore be taken into account when using this driver:

- only use SER2 if no free CPU performance is available.
- do not select too high a total baud rate for all RxD and TxD channels:
- with current modules: sum total = max. approx. 10 kBaud.
- e.g.: 5 X TxD at 1200 Bd or: 1 X RxD + TxD at 4800 Bd.
- The Debug function can be impaired with a higher CPU work-load.

File name: SER2_pp.TDD

INSTALL DEVICE #D, "SER2_pp_xx.TDD", P1,...P7

Note: TIMERA.TDD must be integrated beforehand.

D is a constant, variable or expression of the data type BYTE, WORD, LONG in the range 0...63 and stands for the device number of the driver.

SER2 – serial interfaces through software

- pp** in the file name stands for the position of the send pin (Port, Pin). A table further below in the text shows the location of the pins arising from the selection of the device driver.
- xx** determines the buffer size: R1 = 256 Bytes, K1 = 1 Kbyte, K4 = 4 Kbytes.
- P1...P7** the following table shows the meaning of the parameters P1 to P7:

	leave unchanged	Description of the parameter
P1	0EEH	is a parameter to determine the number of data bits. Value: 1...8
P2	0EEH	is a parameter to determine the parity: 0 = NO 1 = SPACE 2 = Even 3 = Odd 4 = MARK
P3	0EEH	0 = TRUE 1 = INVERS
P4	-	Transmitter Prescaler 0 = no Transmitter present 1 = without Prescaler 2...255 = Prescaler Factor
P5	-	Receive Oversample 0,1,2 = no Receiver present 3...255 = Oversample-Factor
P6	-	Reserved, always 1.
P7	0EEH	Hardware-Handshake Pins: lower 3 Bits: 000 = no Handshake-Pins 001 = CTS-Pin (input, controls send activity) 010 = RTS-Pin (output, shows whether RxD has space in the buffer) 011 = RTS+CTS 100 = Transmitter-Enable f. RS-485 (output, shows whether data in TxD buffer)

SER2 – serial interfaces through software

The device driver uses on to four I/O-pins which can be laid almost at random on the internal I/O pins of the Tiger module. The following table shows which assignments are possible by selecting the suitable driver file:

SER2 – serial interfaces through software

Driver name	TxD (out)	RxD (in)	CTS (in) or TE (out)	RTS (out)
SER2_33_xx.TDD	L33	L34	L35	L70
SER2_34_xx.TDD	L34	L35	L70	L71
SER2_35_xx.TDD	L35	L70	L71	L72
SER2_40_xx.TDD	L40	L42	L33	L34
SER2_42_xx.TDD	L42	L33	L34	L35
SER2_60_xx.TDD	L60	L61	L62	L63
SER2_61_xx.TDD	L61	L62	L63	L64
SER2_62_xx.TDD	L62	L63	L64	L65
SER2_63_xx.TDD	L63	L64	L65	L66
SER2_64_xx.TDD	L64	L65	L66	L67
SER2_65_xx.TDD	L65	L66	L67	L40
SER2_66_xx.TDD	L66	L67	L40	L42
SER2_67_xx.TDD	L67	L40	L42	L33
SER2_70_xx.TDD	L70	L71	L72	L73
SER2_71_xx.TDD	L71	L72	L73	L60
SER2_72_xx.TDD	L72	L73	L60	L61
SER2_73_xx.TDD	L73	L60	L61	L62
SER2_80_xx.TDD	L80	L81	L82	L83
SER2_81_xx.TDD	L81	L82	L83	L84
SER2_82_xx.TDD	L82	L83	L84	L85
SER2_83_xx.TDD	L83	L84	L85	L86
SER2_84_xx.TDD	L84	L85	L86	L87
SER2_85_xx.TDD	L85	L86	L87	L70
SER2_86_xx.TDD	L86	L87	L70	L71
SER2_87_xx.TDD	L87	L70	L71	L72
SER2_74_xx.TD2	L74	L75	L76	L77
SER2_356_xx.TD2	L35	L36	L37	L80
SER2_36_xx.TD2	L36	L37	L80	L81
SER2_37_xx.TD2	L37	L80	L81	L82
SER2_9592_xx.TD2	L95	L92	L40	L42
SER2_7236_xx.TDD	L72	L73	L36	L37

Both incoming and sent data will be buffered in individual buffers with a size depending on the driver version used:

Driver name	Size of buffers
SER2_xx_R1.TDD	256 bytes
SER2_xx_K1.TDD	1024 bytes
SER2_xx_K4.TDD	4096 bytes

Size, level or remaining space of the input and output buffer as well as the driver version can be inquired with the User-Function codes.

User-Function-Codes of the SER2_pp_xx.TDD

User-Function-Codes for inquiries (instruction GET):

No	Symbol Prefix UFCI_	Description
1	UFCI_IBU_FILL	No. of bytes in input buffer (Byte)
2	UFCI_IBU_FREE	Free space in input buffer (Byte)
3	UFCI_IBU_VOL	Size of input buffer (Byte)
33	UFCI_OBU_FILL	Number of bytes in output buffer (Byte)
34	UFCI_OBU_FREE	Free space in output buffer (Byte)
35	UFCI_OBU_VOL	Size of output buffer (Byte)
65	UFCI_LAST_ERRC	Last error code
99	UFCI_DEV_VERS	Driver version

If there is not enough space in the output buffer and you nevertheless wish to output the instruction PUT or Print (and thus the complete task) waits until space once again becomes free in the buffer.

Example: inquire the level of the output buffer to determine whether there is enough space for the output:

```
GET #2, #0, #UFCI_OBU_FILL, 0, wVarFill
IF wVarFill > (LEN(A$)+2) THEN      ' A$ + CR + LF
  PRINT #2, #0, A$
ENDIF
```

SER2 – serial interfaces through software

User-Function-Codes for the instruction PUT following command:

No	Symbol Prefix: UFCO_	Description
1	UFCO_IBU_ERASE	Delete input buffer
33	UFCO_OBU_ERASE	Delete output buffer
94	UFCO_SET_SERIAL	set serial parameters
128	UFCO_SET_ISEP	set limiter characters for instruction INPUT
129	UFCO_RES_ISEP	delete limiter characters for INPUT

Example: set new parameter on serial channel. The parameters will be output in the same way as in the INSTALL line, but only the first 5 parameters are used here:

```
' data,par,inv,TxPre,RxOvs  
PUT #2,#0, #UFCO_SET_SERIAL, 8, 3, 1, 3, 3
```

COMMA and RETURN are regarded as separator characters by default for the instruction INPUT. The separator characters can be changed using the User-Function-Code UFCO_SET_ISEP. Before setting new characters the already set characters can be deleted. The characters to be set or deleted are specified as code areas:

PUT #D, #C, UFCO_SET_ISEP, *Startcode*, *Endcode*, *Startcode*, *Endcode*

! If you delete the standard separators without setting new ones an INPUT instruction will only be terminated when the Input buffer is full.

Example: set new separator LINE-FEED for the instruction input on the serial channel 0:

```
PUT #2,#0, #UFCO_RES_ISEP, 0, 255      ' delete all separators  
PUT #2,#0, #UFCO_SET_ISEP, 10, 10      ' set Line-Feed as separator
```

Example: set all control characters as well as characters as of 7Fh as separator characters for the instruction input on the serial channel 0:

```
PUT #2, #0, #UFCO_RES_ISEP, 0, 255      ' delete all separators  
                                ' set new code area as separators  
PUT #2, #0, #UFCO_SET_ISEP, 0, 31, 127, 255
```

SER2 – serial interfaces through software

Example: delete comma as separator character for the instruction input on the serial channel 0:

```
PUT #2, #0, #UFCO_RES_ISEP, 2ch, 2ch    ' delete comma as separator
\ oder
PUT #2, #0, #UFCO_RES_ISEP, ',,', '      ' delete comma as separator
```

A further example:

```
PUT #1, #0, #UFCI_SET_ISEP, 'acXZ55'
' set as INPUT separators the following characters:
'           a, b, c, X, Y, Z, 5
```

Example: produce echo on serial channel 0:

```
PUT #2, #0, #UFCO_SER_ECHO, YES
```

SER2 – serial interfaces through software

The example program sends on pin L80 (TxD) and receives on pin L81 (RxD).
Connect both pins.

Program example:

```
'-----
'Name: SER2.TIG
'sends characters, and displays received characters
'connect pins L80 (TxD) and L81 (RxD)
'-----
user_var_strict          'variables must be declared
#include DEFINE A.INC     'general defines
#include UFUNC3.INC       'definitions of user function codes

TASK MAIN
  BYTE I
  STRING A$

  '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
  INSTALL_DEVICE #TA, "TIMERA.TDD", 2, 173 '3612Hz for 1200baud
  INSTALL_DEVICE #SER2, "SER2_80.TDD", &
    8, & 'data bits
    0, & 'parity 0=none
    0, & 'invert 0=true, 1=invers
    3, & 'tx prescaler
    3, & 'rx oversample
    1, & 'reserved, always 1
    0 'handshake, 0=none

  PUT #SER2, "hello world<13><10>" 'output with PUT
  PRINT #SER2, "again hello world" 'output with PRINT
  FOR I = 0 TO 0 STEP 0
    GET #SER2, 1, A$ 'read received characters
    PRINT #LCD, A$; 'show on LCD
  NEXT
END
```

SER2 – serial interfaces through software

The following example is useful for experiments since individual characters are sent and received characters shown at the press of a key.

Program example:

```
'-----
'Name: SER2A.TIG
'reads keyboard of Plug & Play Lab
'sends the chars on SER2, and displays chars
'received from SER2 on the LCD
'connect pins L80 (TxD) and L81 (RxD)
'-----
user_var_strict          'variables must be declared
#include DEFINE_A.INC      'general defines
#include UFUNC3.INC        'definitions of user function codes
#include KEYB_PP.INC       'for keyboard of Plug & Play Lab

TASK MAIN
  BYTE ever
  STRING key$, s$          'key and serial character

  'install LCD-driver (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
  'install LCD-driver (TINY-Tiger)
  'INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  install_device #TA, "TIMER.A.TDD", 2, 173 '3612Hz for 1200baud

  INSTALL_DEVICE #SER2, "SER2_80.TDD", &
    8, & 'data bits
    0, & 'parity 0=none
    0, & 'invert 0=true, 1=invers
    3, & 'tx prescaler
    3, & 'rx oversample
    1, & 'reserved, always 1
    0 'handshake, 0=none

  call init_keyb ( LCD )          'init keyboard driver

  for ever = 0 to 0 step 0        'endless loop
    get #SER2, 1, s$              'read received characters
    if s$ <> "" then              'if serial char there
      print #LCD, asc(s$);        'show on LCD as ASCII code
    endif
    get #LCD, 1, key$             'read keyboard
    if key$ <> "" then            'if key there
      put #SER2, key$             'send on soft serial port
    endif
    wait_duration 50              'loop speed
  next
END
```

Documentation History

Version of Documentation	Version of SER2	Description / Changes
001	1.00k	- first version
002	1.00k	- SER2_9592
003	1.00k	- SER2_7236_xx.TDD
004	1.00k	- Some corrections, buffer size table added