The Embedded I/O Company

TEWS TECHNOLOGIES

TIP812-SW-42

VxWorks Device Driver

SERCOS IP

Version 1.0

User Manual

Issue 1.1

September 2003

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek / Germany
Phone: +49-(0)4101-4058-0
Fax: +49-(0)4101-4058-19
e-mail: info@tews.com
www.tews.com

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TIP812-SW-42

SERCOS IP

VxWorks Device Driver

Issue	Description	Date
1.0	First Issue	March 1996
1.1	General Revision	September 2003

Table of Content

1 Introduction

The TIP812-SW-42 VxWorks device driver allows the operation of the TIP812 SERCOS IP as a bus master.

The TIP821 device driver allows:

- initializing the hardware

- setting up the bus timing

- starting up the SERCOS ring through it’s five phases

- transmitting and receiving data

2 Installation

The software is delivered on a 3½" HD diskette.

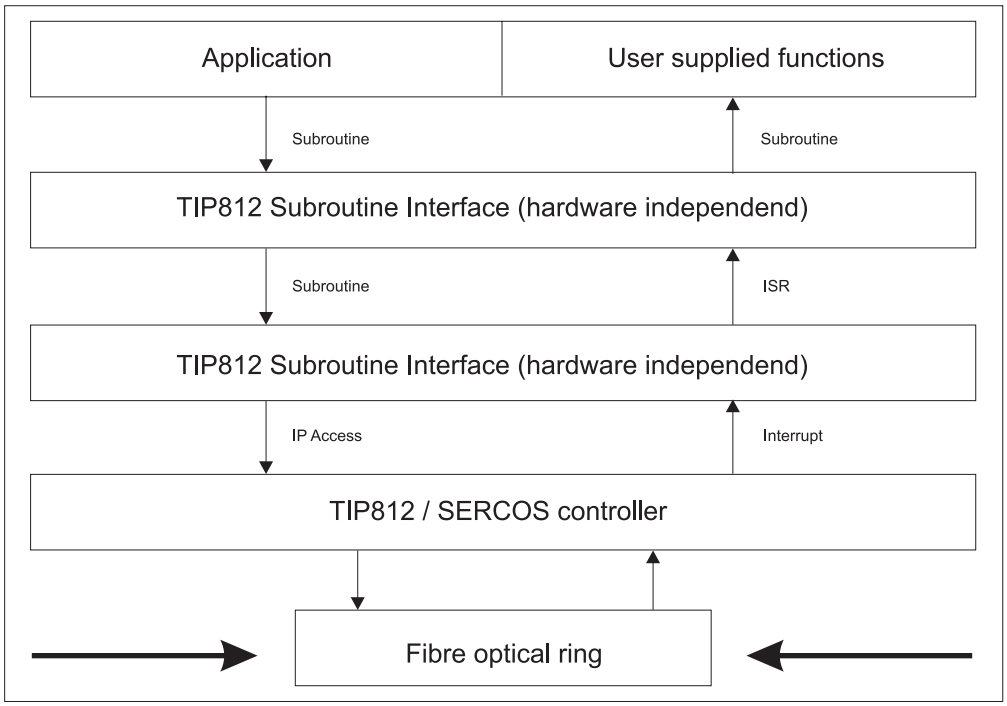
Following files are located on the diskette:

t812drv.c	TIP812 Software Interface
t812drv.h	TIP812 Header of the Software Interfaces
t812_scd.c	TIP812 Hardware dependent functions
t812_scd.h	TIP812 Header for hardware dependent functions
t812_usr.h	TIP812 User definitions
ipchip.h	TIP812 IPIC programming model definitions
sercchip.h	TIP812 SERCON410 programming model definitions
t812main.c	TIP812 Main function of example
t812_tst.c	TIP812 Subfunctions of example
t812_tst.h	TIP812 Header for example

For installation the files have to be copied to the desired target directory.

3 Software Structure

The device driver requires a level structure. The highest level is build by the application followed by the hardware independent functions of the device driver and the hardware dependent functions. The levels below are built by the controller and the other hardware components.



The data flow between the application and the controller is handled over the device driver. Data exchange between the application and the device driver is handled with data structures where the information is placed in. These data structures are described in the chapter *Data Structures* below. Another data flow is realized over the parameters which are delivered by the function calls. They are described in the chapter *Control Functions for TIP812*.

There are three kinds of data which can be transmitted or received. First the service data which are only send if requested. They are sent in a data container and the length is undefined. Second there are cyclic data which are received from the drives. They are sent in ATs (a slot the time cycle). An AT has always the same length and contains always the same data type. The last data type is send from the master to the drives. They are transmitted in the MDT (Master Data Telegram). The MDT is a special time slot in the cycle. It contains data for all dives. The MDT contains always the same data and has always the same length.

4 Control Functions for TIP812

This chapter describes the functions which are used to set and reset the registers of the TIP812 module and to start up a SERCOS ring connection through the five phases.

4.1 Hardware Dependent Functions

These functions will be called by the device driver.

4.1.1 Init_SC_sercon_int1()

NAME

Init_SC_sercon_int1() – prepares the controller for use of INT1 channel.

SYNOPSIS

void Init_SC_sercon_int1 (
int RingNumber)

DESCRIPTION

This function installs prepares the SERCON controller for the use of interrupts.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

EXAMPLE

```
/*-----------------------------------------------------------------------
  Prepare Ring 0 for interrupts
  -----------------------------------------------------------------------*/
Init_SC_sercon_int1 (RING0);
…
```

RETURNS

No return values

4.1.2 Read_SC_sercon_mem()

NAME

Read_SC_sercon_mem() – reads a data block from the DPR of the controller

SYNOPSIS

```
int Read_SC_sercon_mem (
    int         RingNumber,
    WORD        MemOffset,
    int         WordCount,
    WORD*       PData)
```

DESCRIPTION

This function reads a data block from the controllers DPR.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

The parameter **MemOffset** specifies the controller internal offset in the DPR and so the start address of data block which should be read.

The **WordCount** parameter is decisive for the size of the read data block.

PData points to the buffer where the read data should be written to.

Other Information

This function reads words from the controllers DPR and so the buffer size is selected in words. The function returns after the full copy of the buffer.

EXAMPLE

```
…
int       error;
WORD      buffer[16];
…
/*-------------------------------------------------------------------------
  Read 16 Words from controllers DPR address 30
  -------------------------------------------------------------------------*/
error = Read_SC_sercon_mem(  RINGO,
                             30,
                             16,
                             buffer)
…
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.1.3 Write_SC_sercon_mem()

NAME

Write_SC_sercon_mem() – writes a data block to the DPR of the controller

SYNOPSIS

int Write_SC_sercon_mem (

int	RingNumber,
WORD	MemOffset,
int	WordCount,
WORD*	PData)

DESCRIPTION

This function writes a data block to the controllers DPR.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

The parameter **MemOffset** specifies the controller internal offset in the DPR and so the start address of the data block where the data should be written to.

The **WordCount** parameter is decisive for the size of the write data block.

PData points to the data buffer where the data is placed.

Other information

This function writes words to the controllers DPR and so the buffer size is selected in words. The function returns after the full copy of the buffer.

EXAMPLE

```
…
int       error;
WORD      buffer[16];
…
/*-----------------------------------------------------------------------
  Fill data buffer with the data which shall be written
  -----------------------------------------------------------------------*/
…
/*-----------------------------------------------------------------------
  Write 16 Words to controllers DPR address 20
  -----------------------------------------------------------------------*/

error = Read_SC_sercon_mem(  RINGO,
                             20,
                             16,
                             buffer)
…
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.1.4 Read_SC_sercon_reg()

NAME

Read_SC_sercon_reg() – reads a 16 bit value from the register area of the controller

SYNOPSIS

int Read_SC_sercon_reg (

int	RingNumber,
int	RegOffset,
WORD*	Value)

DESCRIPTION

This function reads a 16 bit value from the controllers register area.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

The parameter **RegOffset** specifies the controllers register address of the register which should be set.

The register value will be returned in **Value**.

EXAMPLE

```
…
int       error;
WORD      value;
…
/*-----------------------------------------------------------------------
  Read from register address 16
  -----------------------------------------------------------------------*/
error = Write_SC_sercon_mem(  RINGO,
                              16,
                              &value)
…
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.1.5 Write_SC_sercon_reg()

NAME

Write_SC_sercon_reg() – writes a 16 bit value to the specified register of the controller

SYNOPSIS

```
int Write_SC_sercon_reg (
            int         RingNumber,
            int         RegOffset,
            WORD        Value)
```

DESCRIPTION

This function writes a 16 bit value to a controllers register.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

The parameter **RegOffset** specifies the controllers register address.

The value which shall be set is specified in parameter **Value**.

EXAMPLE

```
…
int           error;
…
/*-----------------------------------------------------------------------
  Set register 12 to 0
  -----------------------------------------------------------------------*/
error = Read_SC_sercon_mem(  RINGO,
                             12,
                             0)
…
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.2 User Supplied Functions

These functions have to be supported by the user. These functions are application dependent and can't be covered by default subroutines.

4.2.1 SC_application_data_info()

NAME

SC_application_data_info() – returns the length parameters of an ident number data block.

SYNOPSIS

```
int SC_application_data_info (
        int     RingNumber,
        int     DriveNumber,
        int     IdentNumber,
        int     ElementNumber)
```

DESCRIPTION

For existing data blocks with the specified ident number the function returns the number of data words, if it is constant or it indicates that the data block length is variable.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

The parameter **DriveNumber** specifies the drive.

The **IdentNumber** selects the data block which length is returned.

The parameter **ElementNumber** selects the data element which length shall be returned.

RETURNS

== 0 if no data block exists for the specified IdentNumber

< 0 if the data length is not constant (the first word contains the actual length and the second word contains the maximal length of the following data).

> 0 if the length is constant (the return value contains the data length. Possible values are 1 and 2).

4.2.2 Transfer_SC_application_data()

NAME

Transfer_SC_application_data () – exchanges service data between master and drive.

SYNOPSIS

```
int Transfer_SC_application_data (
        int         RingNumber,
        int         DriveNumber,
        int         IdentNumber,
        int         ElementNumber,
        int         Direction,
        int         DataOffset,
        WORD*       PBuffer,
        int*        DataLength,
        int*        More Data)
```

DESCRIPTION

This function is used for transmitting service data between the master and the drive. The data length is variable.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

The parameter **DriveNumber** specifies the drive.

The **IdentNumber** selects the data block type.

The parameter **ElementNumber** selects the data element.

The **Direction** selects the transmission direction. Allowed values are *MASTER_TO_DRIVE* and *DRIVE_TO_MASTER*.

The parameter **DataOffset** specifies the number of data words of the selected element which has been exchanged already.

The **pBuffer** parameter points to the data buffer.

- The data should be copied to the buffer if direction selects a master to drive transmission.
- The data should be copied from the buffer if direction selects a drive to master transmission.

The **DataLength** parameter specifies the length of the data buffer.

- If a master to drive communication is selected, the value specifies the maximum length of the data buffer and the function has to change the value before returning to the real data length.
- If a drive to master communication is selected this parameter specifies the real data length.

The **MoreData** parameter indicates if this block is the last block for the selected data element. If the value is zero, it means that the actual block is the last one for the data element. If the value is greater zero it indicates that some more blocks are needed for a complete transmission.

RETURNS

== 0 if function execution is successful or an error number.

4.3 Hardware Independent Functions

4.3.1 Init_SCMaster()

NAME

Init_SCMaster() – initializes TIP812 module and the software as master.

SYNOPSIS

SYNOPSIS

```
int Init_SCMaster (
        int         RingNumber,
        WORD*       register_adr,
        WORD*       dpr_adr,
        int         irq_level)
```

DESCRIPTION

This function initializes the TIP812 SERCOS module and sets up the data structure of the device driver. It also sets the SERCOS interrupt. This function connects the specified TIP812 module to the specified RingNumber. This function must be called before any other function for the ring is called.

Parameter

The **RingNumber** parameter specifies the ring. This number will identify the ring.

The parameter **register_adr** sets the access address for IP register address.

The parameter **dpr_adr** sets the access address for the IP memory.

The value of **irq_level** defines on which level the interrupts are routed.

Other Information

Before calling the Init_SCMaster function the user must set up some values in the data structure of the T_UserRingInfo type. The following values have to be set up:

➔ for every ring:

- PUserRingInfo[RingNumber] ->AtCount = …;
- PUserRingInfo[RingNumber] ->Datarate = …;

➔ for every drive in every ring:

- PUserRingInfo[RingNumber] -> PAtUserData[DriveNumber] -> ATAddress = …;
- PUserRingInfo[RingNumber] -> PAtUserData[DriveNumber] ->CyclicDataCount = …;
- PUserRingInfo[RingNumber] -> PMdtUserData[DriveNumber] ->CyclicDataCount = …;

```
EXAMPLE
#define IP_SLOT_C_REG   0xFFF58200
#define IP_SLOT_C_MEM   0xFFF30000
#define IP_SLOT_C_IRQ        3
…
int  error, i;
…
/*-----------------------------------------------------------------------
  Create T_UserRingInfo and set the values
  -----------------------------------------------------------------------*/
p_UserRingInfo[0] = malloc(sizeof(T_UserRingInfo));
/* set number of ATs in ring */
/* set rings data rate */

/* for every AT in ring */
for(i=0; i<…; i++)
{
         /* set ATaddress of AT[i] */
         /* set the size of cyclic data for AT[i] */
         /* set the size of cyclic data of MDT for AT[i] */
}
/*-----------------------------------------------------------------------
  Initialize the TIP812 module and software (Ring number 0 at IP slot C)
  -----------------------------------------------------------------------*/
error = Init_SCMASTER(  RING0,             /* RingNumber */
                        IP_SLOT_C_REG,     /* Register area of IP */
                        IP_SLOT_C_MEM,     /* Memory area of IP */
                        IP_SLOT_C_RRQ,     /* Interrupt level of IP */
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.3.2 Start_SCMaster_phase()

NAME

Start_SCMaster_phase() – starts one of the communication phases (0..4).

SYNOPSIS

```
int Start_SCMaster_phase (
        int         RingNumber,
        int         PhaseNumber)
```

DESCRIPTION

This function tries to start up the communication of the ring to the selected phase. Only allowed phase switches will be accepted.

Parameter

The **RingNumber** parameter identifies the ring.

The parameter **PhaseNumber** selects the phase which shall be started.

Other Information

This function only starts the phase change. Getting information about the actual state of phase change must be read with Get_SCMaster_phase_status. The starting of a new phase always needs a successful completion of the previous phase.

EXAMPLE

```
…
int error;
…
/*-----------------------------------------------------------------------
  Start phase 0 for ring 0
  -----------------------------------------------------------------------*/
error = Start_SCMaster_phase(RING0, PHASE0);
…
/* Wait until Phase change is completed */
…
/*-----------------------------------------------------------------------
  Start phase 1 for ring 0
  -----------------------------------------------------------------------*/
error = Start_SCMaster_phase(RING0, PHASE1);
…
/* Wait until Phase change is completed */
…
/*-----------------------------------------------------------------------
  Start phase 2 for ring 0
  -----------------------------------------------------------------------*/
error = Start_SCMaster_phase(RING0, PHASE2);
…
/* Wait until Phase change is completed */
…
/*-----------------------------------------------------------------------
  Start phase 3 for ring 0
  -----------------------------------------------------------------------*/
error = Start_SCMaster_phase(RING0, PHASE3);
…
/* Wait until Phase change is completed */
…
/*-----------------------------------------------------------------------
  Start phase 4 for ring 0
  -----------------------------------------------------------------------*/
error = Start_SCMaster_phase(RING0, PHASE4);
…
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.3.3 Get_SCMaster_phase_status()

NAME

Get_SCMaster_phase_status() – returns the status of the last phase change

SYNOPSIS

int Get_SCMaster_phase_status (

int RingNumber)

DESCRIPTION

This function returns the actual state of the last started phase change. This function could be called during or after the change of a phase.

Parameter

The **RingNumber** parameter identifies the ring.

EXAMPLE

```
#define TIMEOUT_COUNT 0x5000     /* test max. 5000 times*/
…
int error;
int timeout;
…
/*-----------------------------------------------------------------------
  Start phase 0 for ring 0
  -----------------------------------------------------------------------*/
error = Start_SCMaster_phase(RING0, PHASE0);
…
/*-----------------------------------------------------------------------
  Wait until Phase 0 has been completed, or an error is found, or the
  timeout counter runs out
  -----------------------------------------------------------------------*/
timeout = TIMEOUT_COUNT;
while ((Get_SCMaster_phase_status(RING0)== NOT_READY) && (timeout > 0))
{
         /* count timeout */
         timeout --;
}
…
```

RETURNS

NOT_READY	phase change is processing, no error
READY	phase change completed successfully
other values	an error was found, the error number is returned

4.3.4 Start_SCMaster_drive_connection()

NAME

Start_SCMaster_drive_connection() – starts a direct communication to a drive.

SYNOPSIS

int Start_SCMaster_drive_connection(

int	RingNumber,
int	DriveNumber)

DESCRIPTION

This function tries to start a direct communication between the master and the specified drive.

Parameter

The **RingNumber** parameter identifies the ring.

The parameter **DriveNumber** selects the drive.

Other Information

This function is only usable in phase 1 and phase 2 because only these two phases allow a direct connection. This function is not interruptible. All interrupts have to be disabled. The function start the connecting process but is doesn't wait for completion. The completion must be detected with *Get_SCMaster_connection_status* function.

EXAMPLE

```
…
int  error;
…
/*-------------------------------------------------------------------------
  Phase 2 is active phase
  Connect the master with drive number 2
  -------------------------------------------------------------------------*/
Start_SCMaster_drive_connection (RING0, 2)
…
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.3.5 Get_SCMaster_connection_status()

NAME

Get_SCMaster_connection_status() – gets the state of the connection process.

SYNOPSIS

int Get_SCMaster_connection_status(

int RingNumber)

DESCRIPTION

This function returns the actual state of the last started connection process.

Parameter

The **RingNumber** parameter identifies the ring.

Other Information

This function is only valid when a connection is started with *Start_SCMaster_drive_connection* function. This function should only be used in phase 1 and phase 2 because the *Start_SCMaster_drive_connection* function is only allowed in this two phases.

EXAMPLE

```
#define TIMEOUT_COUNT   5000      /*Test max. 5000 times */
…
int error;
int timeout;
…
/*-------------------------------------------------------------------------
  Phase 2 is active phase
  Connect the master with drive number 2
  -------------------------------------------------------------------------*/
error = Start_SCMaster_drive_connection (RING0, 2)
/*-------------------------------------------------------------------------
  Wait until connection is ok, or timout runs out
  -------------------------------------------------------------------------*/
timeout = TIMEOUT_COUNT;
while ((Get_SCMaster_connection_status(Ring0)== NOT_READY) && (timeout >0
))
{
     timeout --;
}
…
```

RETURNS

NOT_READY	connection process is active, no error
READY	connection process completed successfully, connection is valid
other values	an error was found, the error number is returned

4.3.6 Start_SCMaster_service_transfer()

NAME

Start_SCMaster_service_transfer() – starts the connection process for service data

SYNOPSIS

```
int Start_SCMaster_service_transfer(
        int     RingNumber,
        int     DriveNumber,
        int     IdentNumber,
        int     ElementMask,
        int     Direction)
```

DESCRIPTION

This function starts a connection between the master and the specified drive.

Parameter

The **RingNumber** parameter identifies the ring.

The parameter **DriveNumber** specifies the target drive.

The **IdentNumber** selects which data should be send.

The **ElementMask** parameter shows the data segments which shall be transmitted.

The **Direction** selects if the master shall read or write service data.

Other Information

This function should not be used before phase 2 is reached. A started service transfer can be aborted with the function *Abort_SCMaster_service_transfer* while the transfer status is active. The actual state of the transfer can be read out with *Get_SCMaster_service_status* function.

EXAMPLE

```
int error;
…
/*-----------------------------------------------------------------------
  Receive data from drive 1
  -----------------------------------------------------------------------*/
error = Start_SCMaster_service_transfer (  RING0,
                                           1,
                                           95,
                                           0x80,
                                           DRIVE_TO_MASTER);
…
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.3.7 Abort_SCMaster_service_transfer()

NAME

Abort_SCMaster_service_transfer() – aborts a service transfer between master and specified drive.

SYNOPSIS

int Abort_SCMaster_service_transfer(

int	RingNumber,
int	DriveNumber)

DESCRIPTION

This function aborts a connection between the master and the specified drive.

Parameter

The **RingNumber** parameter identifies the ring.

The parameter **DriveNumber** specifies the target drive.

Other Information

This function quits a service transfer which was previously started with *Abort_SCMaster_service_transfer* function.

EXAMPLE

```
int error;
…
/*-----------------------------------------------------------------------
  Receive data from drive 1
  -----------------------------------------------------------------------*/
error = Start_SCMaster_service_transfer (  RING0,
                                           1,
                                           95,
                                           0x80,
                                           DRIVE_TO_MASTER);
…
error = Abort_SCMaster_service_transfer (Ring0,1);
…
```

RETURNS

Always *NO_ERROR*.

4.3.8 Get_SCMaster_service_status()

NAME

Get_SCMaster_service_status() – get the state of the service transfer.

SYNOPSIS

int Get_SCMaster_service_status(

int	RingNumber,
int	DriveNumber,
WORD*	Error State,
WORD*	CmdReceipt,
int*	CMDChange)

DESCRIPTION

This function returns the actual state of the last started service data transfer.

Parameter

The **RingNumber** parameter identifies the ring.

The parameter **DriveNumber** specifies the target drive.

The parameter **ErrorState** returns the drive service information or the error code *ERROR_NODATA* or *ERROR_SCHSTIMEOUT*.

The parameter **CmdReceipt** holds the data state.

The **CmdChange** parameter presents the command change bit.

Other Information

This function is only valid if a service transfer is started with *Start_SCMaster_service_transfer* function for this drive.

EXAMPLE

```
#define TIMEOUT_COUNT   5000      /* Test max. 5000 times */
…
int       error;
int       timeout;
WORD      ErrorState,
WORD      CmdReceipt,
int       CmdChange)
/*-------------------------------------------------------------------------
  Receive data from drive 1
  -------------------------------------------------------------------------*/
error = Start_SCMaster_service_transfer ( RING0,
                                          1,
                                          95,
                                          0x80,
                                          DRIVE_TO_MASTER);
/*-------------------------------------------------------------------------
  Wait until data transfer is complete
  -------------------------------------------------------------------------*/

timeout = TIMEOUT_COUNT;
while ((Get_SCMaster_service_status(RING0, 1, &ErrorState, &CmdReceipt,
&CmdChange)== NOT_READY) && timeout >0 ))
{
timeout --;
}
…
```

RETURNS

NOT_READY	connection process is active, no error
NOT_READY_BUT_BUSYTIMEOUT	data transfer active but the time limit is passed
READY	connection process completed successfully, connection is valid
other values	an error was found, the error number is returned

4.3.9 Prepare_SCMaster_for_phase3()

NAME

Prepare_SCMaster_for_phase3() – calculates the communication parameter for phase 3 and phase 4.

SYNOPSIS

int Prepare_SCMaster_for_phase3(

int RingNumber)

DESCRIPTION

This function calculates the communication parameter for phase 3 and phase 4. Phase 3 and phase 4 communication cycles are split into time slices. This function calculates all time slices for the selected ring.

Parameter

The **RingNumber** parameter identifies the ring.

Other Information

This function needs the following values filled in the data structure T_UserRingInfo:

➔ filled by user

- PUserRingInfo[RingNumber] -> AtCount = …;
- PUserRingInfo[RingNumber] -> Datarate = …;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] -> Tscyc = …;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->CyclicDataCount=…;
- PUserRingInfo[RingNumber] -> PMdtUserTime[0.. last_drive -1] -> CyclicDataCount=…;

➔ read over service channel

- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->T1min =…;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->T4min =…;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->Tmtsg =…;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->Tmtsy =…;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->Tatmt =…;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->TSlavekennung =…;
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->Tatat =…;

(if more than one drive at slave)

After calculating the values they have to be written to the drives using the service channel.

➔ calculated values:

- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->T1
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->T2
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->T3
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->T4
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->LaengeMasterDaten
- PUserRingInfo[RingNumber] -> PATUserTime[0..last_drive -1] ->OffsetMasterDaten

EXAMPLE

```
…
int  error;
…
/*-----------------------------------------------------------------------
  fill the T_UserRingInfo
  -----------------------------------------------------------------------*/
…
/*-----------------------------------------------------------------------
  Calculate the communication parameters
  -----------------------------------------------------------------------*/
error =  Prepare_SCMaster_for_phase3(Ring0);
…
/*-----------------------------------------------------------------------
  Send the calculated times to the drives and use the service channel
  -----------------------------------------------------------------------*/
```

RETURNS

NO_ERROR if function execution is successful. Any other value if an error is detected.

4.3.10 Handle_SCMaster_Interrupt()

NAME

Handle_SCMaster_Interrupt() – analyzes events (interrupt flags) and activates user supplied functions.

SYNOPSIS

void Handle_SCMaster_interrupt(

int RingNumber)

DESCRIPTION

This function analyzes the interrupt flags of the controller selected by RingNumber. If an event is detected a user supplied function will be called.

Parameter

The **RingNumber** parameter identifies the ring.

Other Information

This function has to be called minimal once per communication cycle if interrupts aren't active. If the interrupts are active the interrupt service will pass this function.

RETURNS

No return code.

5 Data Structures

The data exchange between user program and device driver is handled with the data structure *PUserRingInfo[MAXRING]* of the type *T_UserRingInfo*.* The structure is described in this chapter. There are two kinds of values in this structure, first the values which are read only. They are marked with *RO*. The registers which could be written are marked with *WR*.

```
struct      T_User_RingInfo     *PUserRingInfo[MAXRING]     /* One structure for every ring  */
                                                            /* communication structure for  */
                                                            /*max. MAXAT drives  */
struct      T_UserRingInfo
{
int                             AtCount;                    /* (WR) */
                                                            /* number of drives in the ring */
int                             Datarate;                   /* (WR) - data rate for the ring */
struct      T_AtUserData*       PAtUserData [MAXAT];
struct      T_AtUserTime*       PAtUserTime[MAXAT];
struct      T_MdtUserData*      PMdtUserData[MAXAT];
WORD                            MdtHeaderOffset;            /* (RO) Start of the MDT */
                                                            /* Header in the DPR */
WORD                            EndHeaderOffset;            /* (RO) */
};
/* Communication user parameter for one AT */
struct      T_AtUserData
{
int                             AtAddress;                  /* (WR) – phys. drive address */
int                             CyclicDataCount;            /* (WR) */
                                                            /* length of cyclic data in WORDs */
WORD                            HaederOffset;               /* (RO)  */
                                                            /* Start of the AT header in the DPR */
WORD                            DataContainerOffset;        /* (RO) */
                                                            /* Start of the data area in the DPR */
int                             DataContainerLength;        /* (RO) */
                                                            /* Length of the data area in WORDs
};
```

```
/* Communication parameter for on AT (drive) */
struct      T_AtUserData
{
WORD                    T1min;              /* (WR) */
WORD                    Tatat;              /* (WR) */
WORD                    Tatmt;              /* (WR) */
WORD                    Tmtsy;              /* (WR) */
WORD                    Tmtsg;              /* (WR) */
WORD                    T4min;              /* (WR) */
WORD                    Tscyc;              /* (WR) */
WORD                    Tncyc;              /* (WR) */
WORD                    Slavekennung;       /* (WR) */
WORD                    T1;                 /* (RO) */
WORD                    T2;                 /* (RO) */
WORD                    T3;                 /* (RO) */
WORD                    T4;                 /* (RO) */
WORD                    Laenge Master Daten; /* (RO) – Length of the data set  */
                                            /* for the drive in the MDT in  */
                                            /* BYTEs */
WORD                    OffsetMasterDaten;  /* (RO) – Start of the data set */
                                            /* for the drive in the MDT in */
                                            /* BYTEs */
};
/* Communication parameter for the MDT */
struct      T_MdtUserData
{
int                     CyclicDataCount;    /* (WR) – Length of the cyclic data */
                                            /* (without control and info) */
WORD                    DataContainerOffset; /* (RO) */
                                            /* Start of the data area in the DPR */
Int                     DataContainerLength; /* (RO) */
                                            /* Length of the data area in WORDs */
};
```

6 Definitions (Constant Values)

This chapter describes the values which are defined in the device driver. Using these predefined values avoids the use of forbidden values.

6.1 Baud Rate

(0x01)	BAUD_2M	Baud rate is set to 2Mbit/s
(0x00)	BAUD_4M	Baud rate is set to 4Mbit/s

6.2 State Messages

(0x0000)	READY	The last process completed and no error was found (Example: Phase change -> Phase change completed successfully).
(0x7FFF)	NOT_READY	The last process is still active (Example: Phase change -> Phase change is still in progress).
(0x70FF)	NOT_READY_BUT_BUSYTIMEOUT	The last process has not completed, but an error was found.

6.3 Data Direction

(0x00)	MASTER_TO_DRIVE	Data is send to the drive.
(0x01)	DRIVE_TO_MASTER	Data is received from the drive.

6.4 Phases

(0x00)	PHASE0	Phase is Phase 0.
(0x01)	PHASE1	Phase is Phase 1.
(0x02)	PHASE2	Phase is Phase 2.
(0x03)	PHASE3	Phase is Phase 3.
(0x04)	PHASE4	Phase is Phase 4.
(0x05)	DRIVE_INIT	There is no phase started.

6.5 Ring Numbers

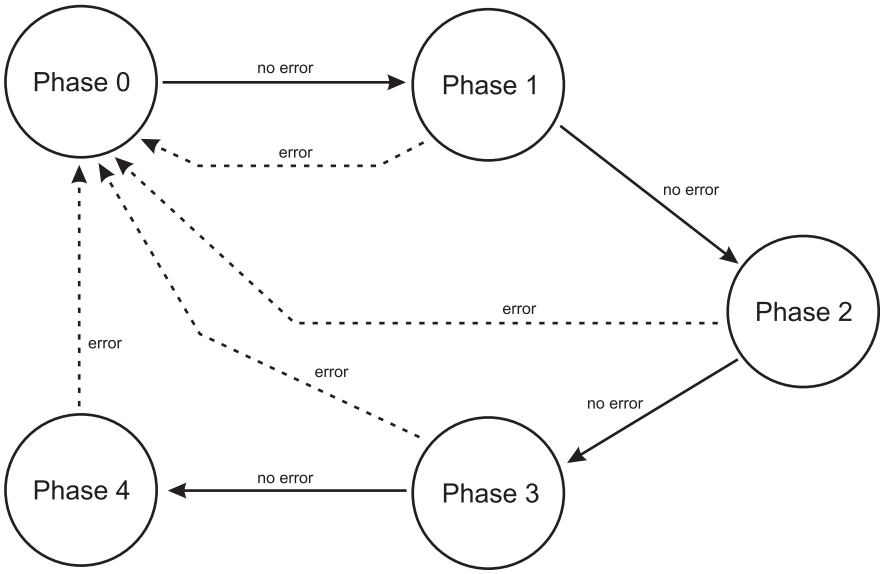
(0x00)	RING0	1st ring
(0x01)	RING1	2nd ring
(0x02)	RING2	3rd ring
(0x03)	RING3	4th ring
(0x04)	RING4	5th ring
(0x05)	RING5	6th ring
(0x06)	RING6	7th ring
(0x07)	RING7	8th ring

7 Start up the Ring

This chapter gives a short description of starting the communication over the ring. After reaching phase 4 the communication over the ring is full available. An example of starting up the ring is delivered with the code of the device driver.

7.1 Error Handling

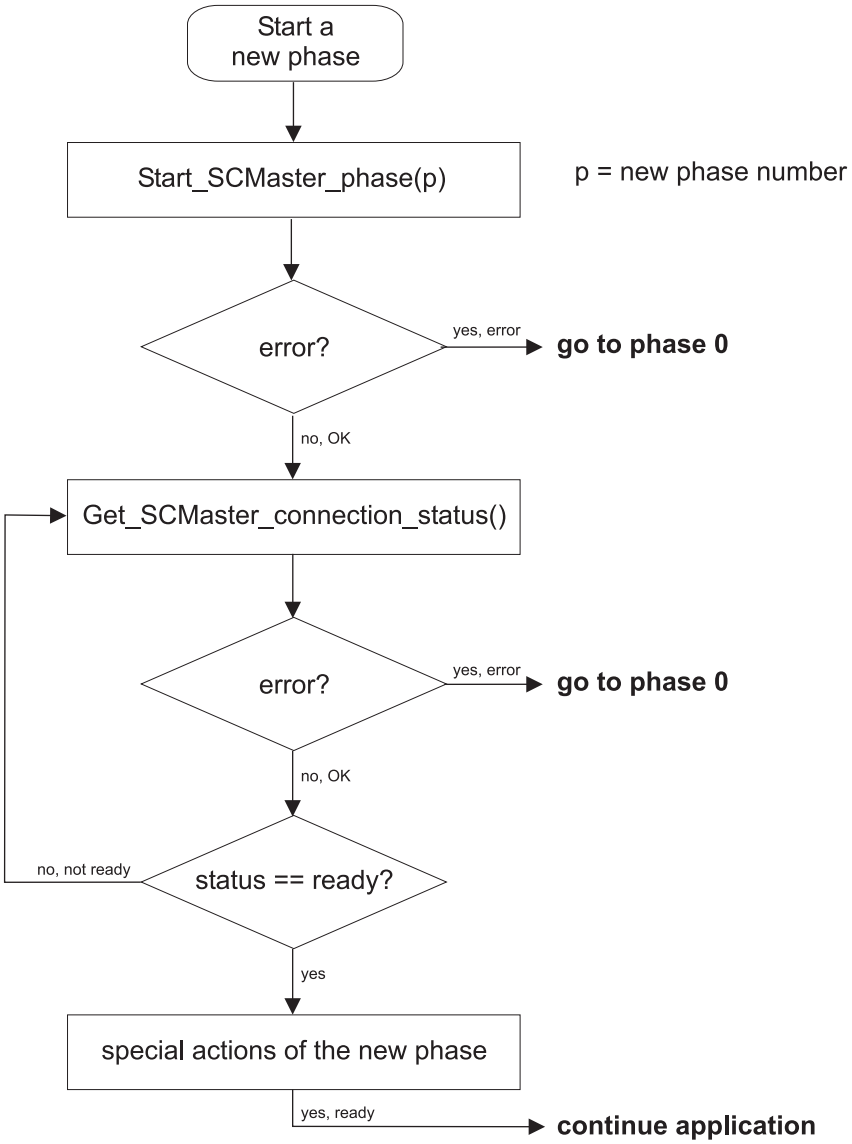
The start up has to start with phase 0 and go over the phases 1, 2 and 3 to phase 4. If any phase detects an error the start up must start again with phase 0.



7.2 Start a New Phase

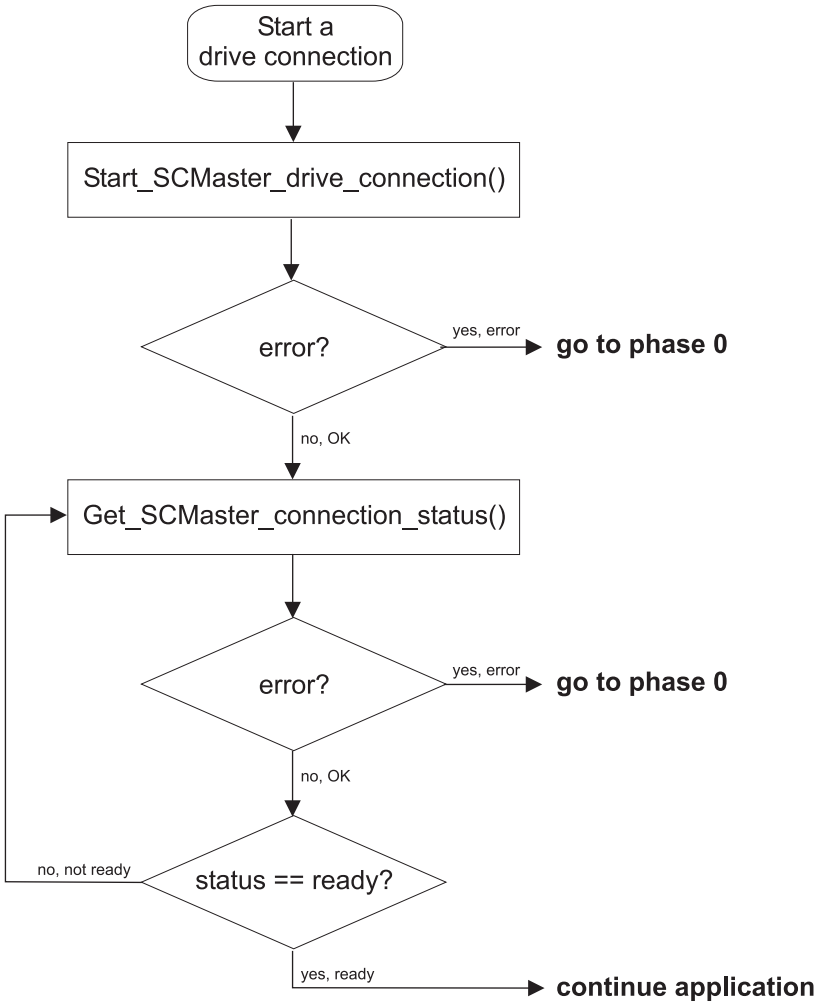
Starting a new phase is divided into three main parts, first the phase switch, second is waiting on phase change completion and third the actions which have to take place in the phase. These actions are defined by the SERCOS specification (Example: in phase 1 the master has to check if all drives are answering over the bus).

The next diagram shows the rough program run for a start of a new phase.



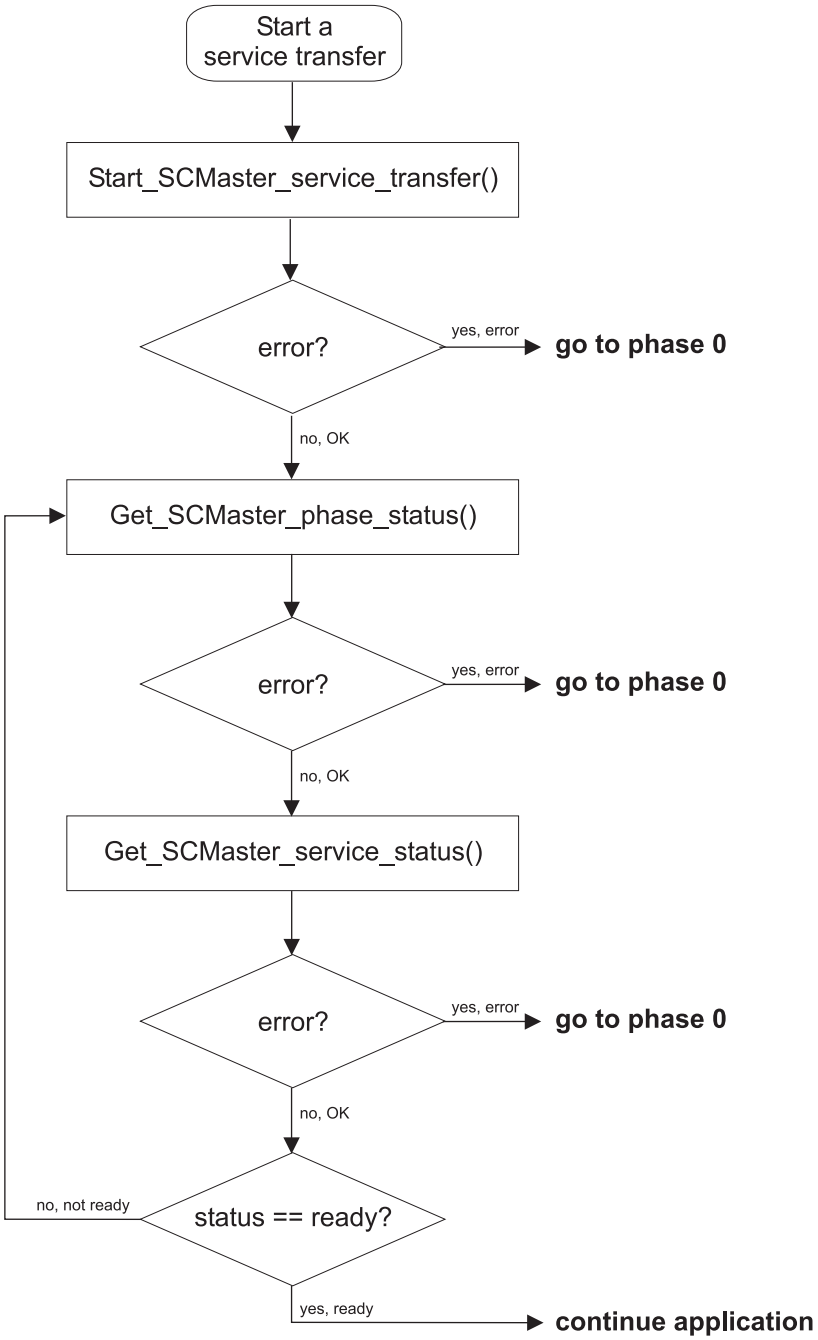
7.3 Start a Drive Connection

Starting a drive connection is only allowed in phase 1 and phase 2. In phase 1 it's used to check if all drives are answering. In phase 2 it is used to exchange the communication parameters. After starting a drive connection the application has to wait until the connection is complete. The following diagram shows the principle of the start of a drive connection.



7.4 Start a Service Transfer

The service transfers can be started when phase 2 or a higher is running. Service is used to send parameters or other data to the selected drive. After starting a service transfer the application must check the connection and the communication, if there is an error or if the communication is complete. The following diagram gives a rough overview.



8 Error Codes

(0x00000001)	ERROR_NOMEMORY	There is not enough memory for internal data structures.
(0x00000002)	ERROR_SETINTVEC	The interrupt vector is missing or not ok.
(0x00000003)	ERROR_HSTIMEOUT	Host time out, no AT has been received during 10 MDTs.
(0x00000004)	ERROR_ATMISS	Two or more ATs are missing.
(0x00000005)	ERROR_WRONGPHASE	The selected phase is not reachable from the actual state.
(0x00000006)	ERROR_WRONGADDRESS	The slave address is not allowed.
(0x00000007)	ERROR_WRONGRING	The selected ring number is not placed in the allowed area.
(0x00000008)	ERROR_WRONGATCOUNT	There are too much slaves in the ring.
(0x00000009)	ERROR_WRONGATNUMBER	False or not allowed slave number.
(0x00000010)	ERROR_DPRAMOVERFLOW	The memory area for cyclic data is smaller than the needed.
(0x00000011)	ERROR_SCNOTINIT	The memory area for data channels is not big enough.
(0x00000012)	ERROR_SCTRANSNOTREADY	The previous started data transmission is not ready.
(0x00000013)	ERROR_SCTRANSNODATA	No data available for data channel.
(0x00000014)	ERROR_CALCULATE_T2	The telegrams don't fit to the telegram timing or the cycle time is to short.
(0x00000015)	ERROR_CALCULATE_T3	The telegrams don't fit to the telegram timing or the cycle time is to short.
(0x00000016)	ERROR_CALCULATE_T4	The telegrams don't fit to the telegram timing or the cycle time is to short.
(0x00000017)	ERROR_CALCULATE_TEND	The telegrams don't fit to the telegram timing or the cycle time is to short.
(0x0000FFFF)	ERROR_DEFAULT	
(0x00000018)	ERROR_INIT	
(0x00000019)	ERROR_SCNODATA	No data available
(0x00000020)	ERROR_SCHSTIMEOUT	Data telegram timed out
(0x08120100)	ERROR_NODEVICE	There is no TIP812 module mounted to the specified address.
(0x08120101)	ERROR_CHIPCOUNT	False ring number
(0x08120102)	ERROR_REGISTERADR	The selected register number is invalid.
(0x08120103)	ERROR_DPROFFSET	The access area is not fully placed in the DPR.
(0x08120105)	ERROR_SETREG	Error setting a register