
**Sleep Mode Handling of the Atmel LIN Transceiver
ATA6663/ATA6664**

ATA6663/ATA6664

Introduction

The Atmel® ATA6663 and Atmel ATA6664 are fully integrated LIN transceivers designed to handle low-speed data communication in vehicles and features three different operating modes: normal mode, sleep mode and fail-safe mode.

This document describes how to switch these LIN Transceivers securely to different operating modes (especially sleep mode), including basic code examples. These code examples assume that the part-specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header file. The code examples are limited to the Atmel AVR® microcontrollers and the IAR® C and GCC® compilers.

1. Operating Modes

Figure 1-1 shows the operating modes available in the Atmel® ATA6663/ATA6664 LIN transceiver as well as the transitions between them. Switching the LIN transceiver to sleep mode is only possible from normal mode, but when in sleep mode the LIN transceiver can be woken up by various events:

- LIN bus (remote wake-up)
- EN pin
- WAKE pin (local wake-up)
- Vs undervoltage

Figure 1-1. Atmel ATA6663/ATA6664 - Operating Modes

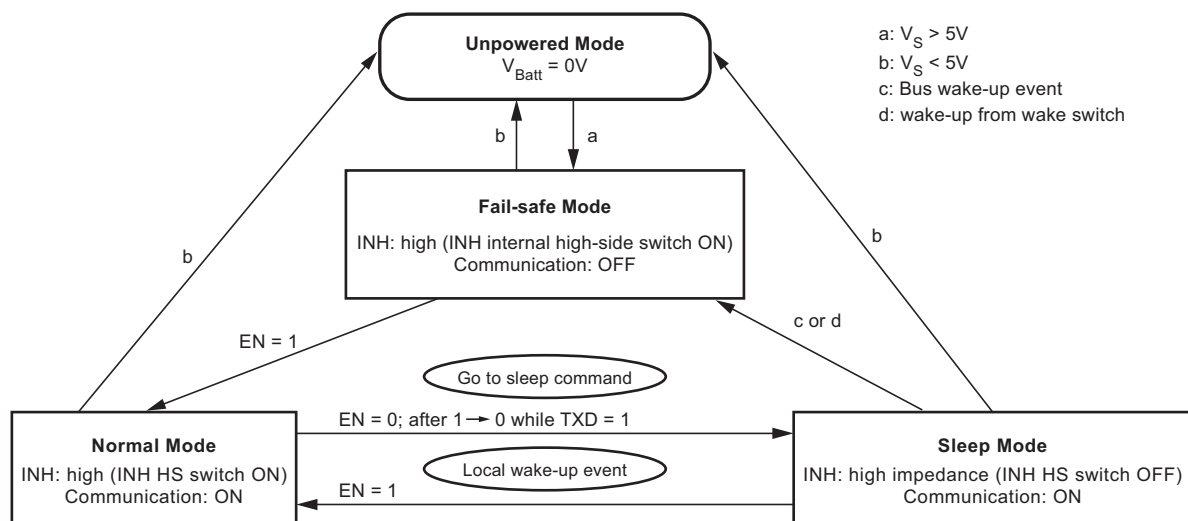


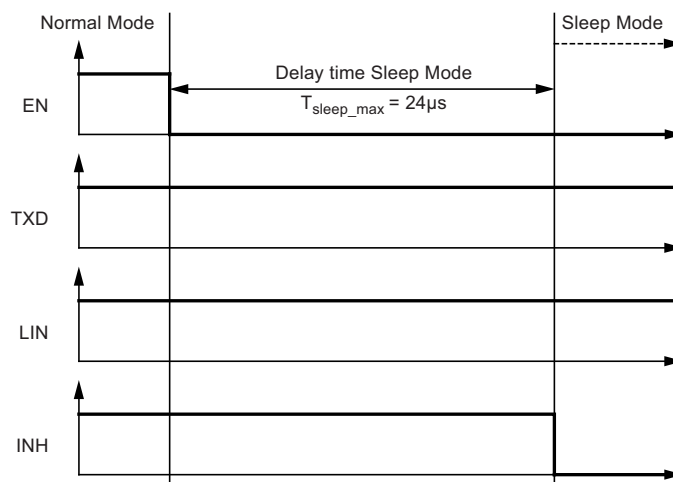
Table 1-1. Operating Modes

Operating Mode	Transceiver	INH	RXD	LIN
Fail-safe	Off	On, except $V_S < 5V$	High, except after wake-up	Recessive
Normal	On	On	LIN-dependent	TXD-dependent
Sleep	Off	Off	High ohmic	Recessive

1.1 Putting Atmel ATA6663/ATA6664 in Sleep Mode

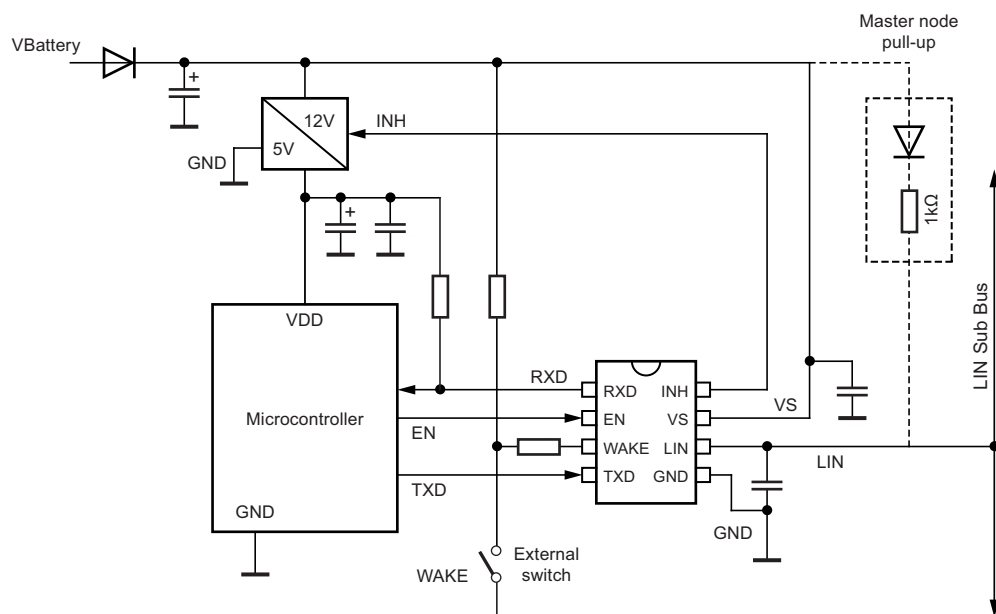
In order to switch the Atmel ATA6663/ATA6664 into sleep mode the EN pin has to be pulled to GND and TXD has to remain at high level. After the time T_{sleep} , beginning directly at the falling edge of the Enable signal, has expired, the device will enter the sleep mode and the INH output will be switched off.

Figure 1-2. Switching to Sleep Mode



A typical application is shown in [Figure 1-2](#) where the INH pin controls an external voltage regulator. After setting the LIN transceiver to sleep mode the INH output switches off the voltage regulator so that the microcontroller is no longer supplied.

Figure 1-3. Typical Application



There are two ways to wake up the module: a remote wake-up via LIN or a local wake-up via the WAKE pin. The device will also wake-up in the case of an undervoltage at the pin VS. In all cases the LIN transceiver enters fail-safe mode, the INH output is switched on and the voltage regulator supplies the microcontroller.

Waking up the LIN transceiver through the EN pin is impossible in this particular application ([Figure 1-2 on page 3](#)) because the microcontroller is not supplied when the LIN transceiver is in sleep mode.

Precautions have to be taken to avoid unwanted deadlocks caused by immediate wake-up after going to sleep or under-voltage when securely switching to sleep mode. [Section 1. "Operating Modes" on page 2](#) provides a detailed description of this behavior and various solutions (hardware and software) for resolving the issue. The basic code example in 2.2 shows the software solution for switching the Atmel® ATA6663/ATA6664 LIN transceiver into sleep mode.

1.2 Basic C Routine for Putting Atmel ATA6663/ATA6664 into Sleep Mode

The Atmel ATA6663/ATA6664 transceiver can be put into sleep mode using a basic C function:

```
unsigned char Switch_ATA6663_64_To_SleepMode (void)
{
    // Before setting the EN pin to low it must be ensured that the TXD line
    // remains high. Consequently, the TXD pin has to be set to output high
    // here at the latest. The output high will be overwritten by the UART as
    // long as the UART transmitter is enabled.
    // Consequently, the UART transmitter must be disabled.
    TXDPIN_high;          // Set the TXD pin @ the microcontroller to output high
    TXDPIN_output;        // Set the TXD pin to output in the DDRx register
    UCSRB &= ~(1<<TXEN); // Disable the UART transmitter

    // Now the actual switching of the ATA6663/64 into the sleep mode takes place
    ENABLEPIN_low;        // Switch EN pin to low; assuming that the
    // microcontroller pin is already set to output
    // From this point the power supply of the microcontroller will be
    // switched off sooner or later. The microcontroller is waiting for this
    // to happen.
    while(1)
    {
        // Active waiting for the supply to be switched off
    };
}
```

1.3 Possible Issues without Using any Safety Precautions

There are two scenarios which could lead to the situation that the connected microcontroller remains powered after having the Atmel ATA6663/ATA6664 switched into sleep mode: immediate wake-up after “going to sleep” or an undervoltage when switching to sleep mode.

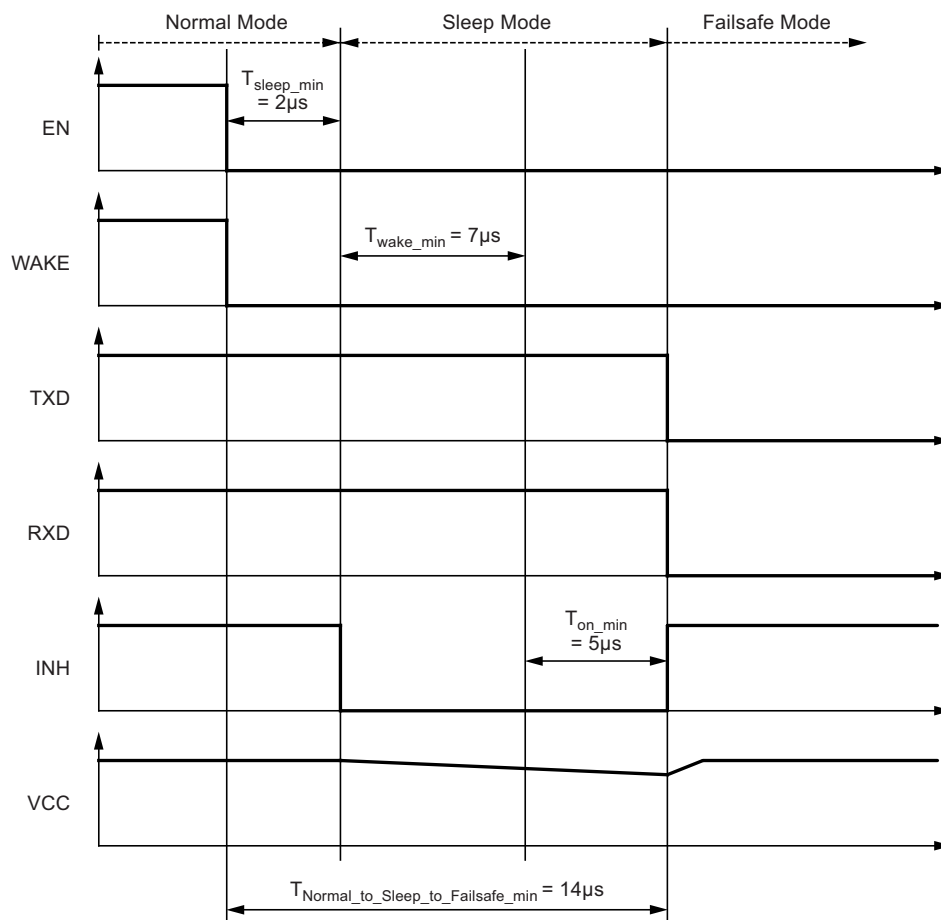
In the basic C routine above the microcontroller waits for the blocking capacitors to be discharged and the supply voltage to end. But in case of immediate wake-up after “going to sleep” or undervoltage when switching to sleep mode, this never occurs and the complete module will end in a deadlock.

1.4 Immediate Wake-up after “Going to Sleep”

A very short low phase at the INH pin in combination with the blocking capacitors at the output of the voltage regulator is the basis for the first scenario.

One juncture where this can happen is the presence of a wake signal at the same time or slightly later than the falling edge of the EN signal. In this case the device will wake up again after the INH pin has been at low level for only 12µs in the worst case. The relevant signals during this scenario are illustrated in the following graph.

Figure 1-4. Immediate Wake-up after “Going to Sleep”



Depending on the value of the blocking capacitors at Vcc and the current consumption of the microcontroller, the microcontroller may remain powered during the 12µs.

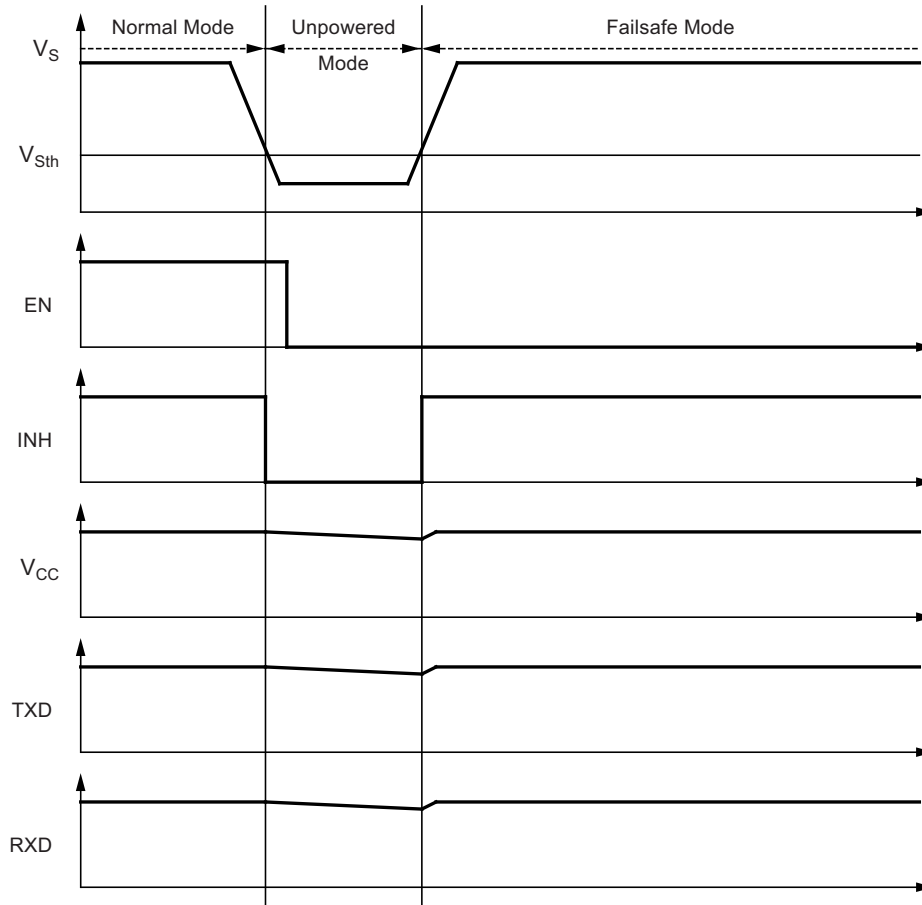
1.5 Undervoltage When Switching to Sleep Mode

The second scenario that might lead to a deadlock of the module is the case of supply voltages below +5V. In this case the Atmel® ATA6663/ATA6664 switches automatically into the unpowered mode without signaling the mode change to the microcontroller. In general, the unpowered mode is the same as the fail-safe mode, but the Atmel ATA6663/ATA6664 does not change its state according to the EN pin.

If the microcontroller does not change the state of the EN pin (the EN pin remains high) during this period, then the Atmel® ATA6663/ATA6664 will automatically re-enter normal mode after the battery voltage has recovered. In compliance with the LIN specification, no data transmission via LIN is possible during the undervoltage period.

If the Atmel ATA6663/ATA6664 detects undervoltage and it has changed to the unpowered mode, any further mode change via the EN pin will not lead to any action. The INH output is switched off during the unpowered mode. The microcontroller remains active for a certain time due to the blocking capacitors at the Vcc pin of the microcontroller. As the microcontroller has no information about the Atmel ATA6663/ATA6664 being in the unpowered mode it may attempt to switch the device into sleep mode at or directly after this very point in time. This scenario is depicted in [Figure 1-5 on page 6](#).

Figure 1-5. Undervoltage When Switching to Sleep Mode



After undervoltage at V_S disappears, the Atmel ATA6663/ATA6664 switches from unpowered mode to fail-safe mode and as a result the INH output switches on again. The Atmel ATA6663/ATA6664 will stay in fail-safe mode, as the EN pin is still low. But the microcontroller expects the Atmel ATA6663/ATA6664 to have entered sleep mode and is caught in the active waiting loop waiting for its power supply to be switched off. Consequently, it does not even consider putting the Atmel ATA6663/ATA6664 into normal mode via the EN pin, and the module remains in a deadlock, isolated from LIN communication.

2. How to Avoid a Potential Deadlock

Common to all of the possibilities for a deadlock scenario described is that the microcontroller remains powered while the Atmel® ATA6663/ATA6664 has entered and already left sleep mode and remains in fail-safe mode. The power supply of the module has been switched off for only a few microseconds, which, due to the blocking capacitors, will not always lead to an undervoltage reset of the microcontroller. As the microcontroller is in an endless loop expecting either a reset signal or its supply voltage to disappear, which will never occur, the module is in a deadlock.

This scenario can be avoided by either a hardware solution or a software solution.

2.1 Hardware Solutions

2.1.1 External Watchdog

When the microcontroller stays alive but is only waiting for its power supply to disappear, an external watchdog will reset the microcontroller. The precondition for this is, of course, that the microcontroller stops serving the watchdog before entering the waiting loop.

2.1.2 External Voltage Regulator with Reset Output

As the Atmel ATA6663/ATA6664 indicates all of the scenarios mentioned by switching off the INH output, which will normally lead to an undervoltage reset of the connected microcontroller, the voltage regulator being used should react to this INH signal quickly and directly. This means the voltage regulator should be able to generate a reset signal for the microcontroller as long as its INH input is low. Switching off the output voltage might not be sufficient, as the remaining output voltage stored in the blocking capacitors can be sufficient to keep the microcontroller alive.

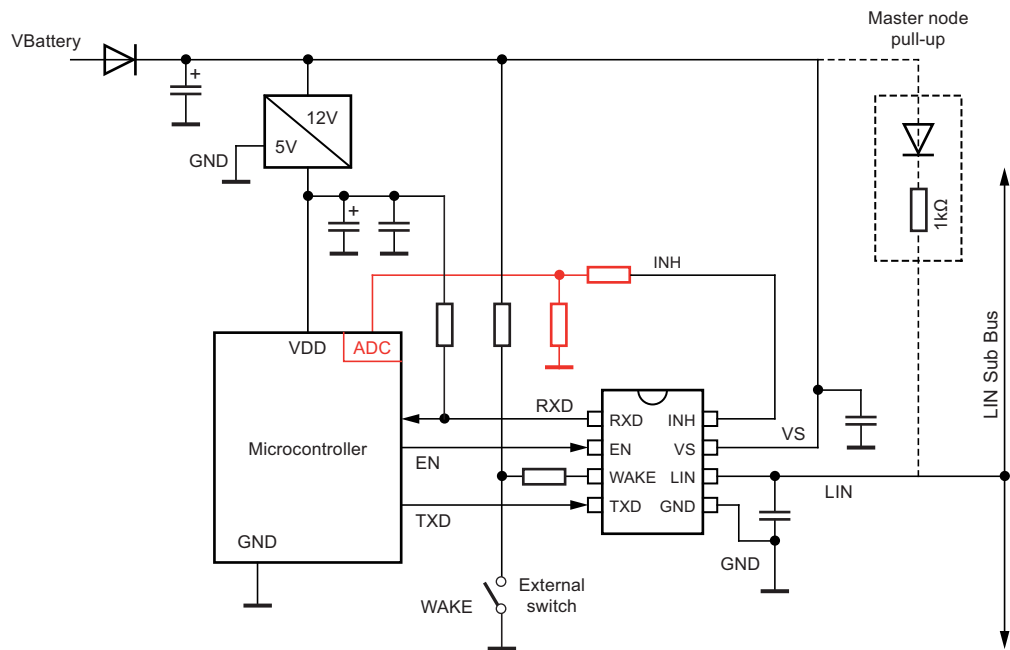
2.1.3 INH Monitoring

If the voltage regulator has no ability to generate an INH-dependent reset signal, then the microcontroller itself can take over this task. The easiest solution for doing this is to connect the INH pin to a dedicated microcontroller I/O pin to monitor the logic level of the INH pin. The microcontroller can then generate an internal reset according to the INH status.

But as the INH pin is a battery-related output, special care must be taken when connecting this pin to the microcontroller. With an additional voltage divider between the INH output pin and ground the logical level of the INH output can easily be read out and the voltage level at the microcontroller input is limited to noncritical values. With this voltage divider it is also possible to monitor the battery voltage via the INH output pin. In this case the voltage divider has to be connected to an A/D converter input of the microcontroller as shown in [Figure 2-1 on page 8](#) in red color.

The current consumption of the module will not be increased in sleep mode, as the INH will be switched off in sleep mode, so that the voltage divider is not active in this mode.

Figure 2-1. Monitoring the Supply Voltage via the INH Pin



2.2 Software Solution

In addition to hardware based solutions, a software routine also makes it possible to identify when the Atmel® ATA6663/ATA6664 is not in the expected sleep mode. When the microcontroller expects its power supply to disappear, an additional time-out can be implemented for canceling the waiting loop after the time-out has expired. The necessary time-out time depends mainly on the values of the blocking capacitors and the output current of the voltage regulator. The necessary minimum time-out time can easily be calculated using the following equations.

$$Q = C \times U \quad (1-1)$$

$$Q = I \times t \quad (1-2)$$

$$C \times U = I \times t \quad (1-3) \text{ using (1-1) and (1-2)}$$

$$\Rightarrow t_{\min} = C \times U / I \quad (1-4)$$

C: The total capacitance of all blocking capacitors at the power supply of the microcontroller

I: Total output current of the voltage regulator the microcontroller is connected to.

U: Voltage difference between the nominal output voltage and the undervoltage reset threshold (if there is one) or the brownout detection threshold.

The resulting time is the minimum time for the blocking capacitors to be discharged. Because the current output is almost never constant, the calculated time should be at least doubled in order to be sure that enough time has elapsed before leaving the endless loop.

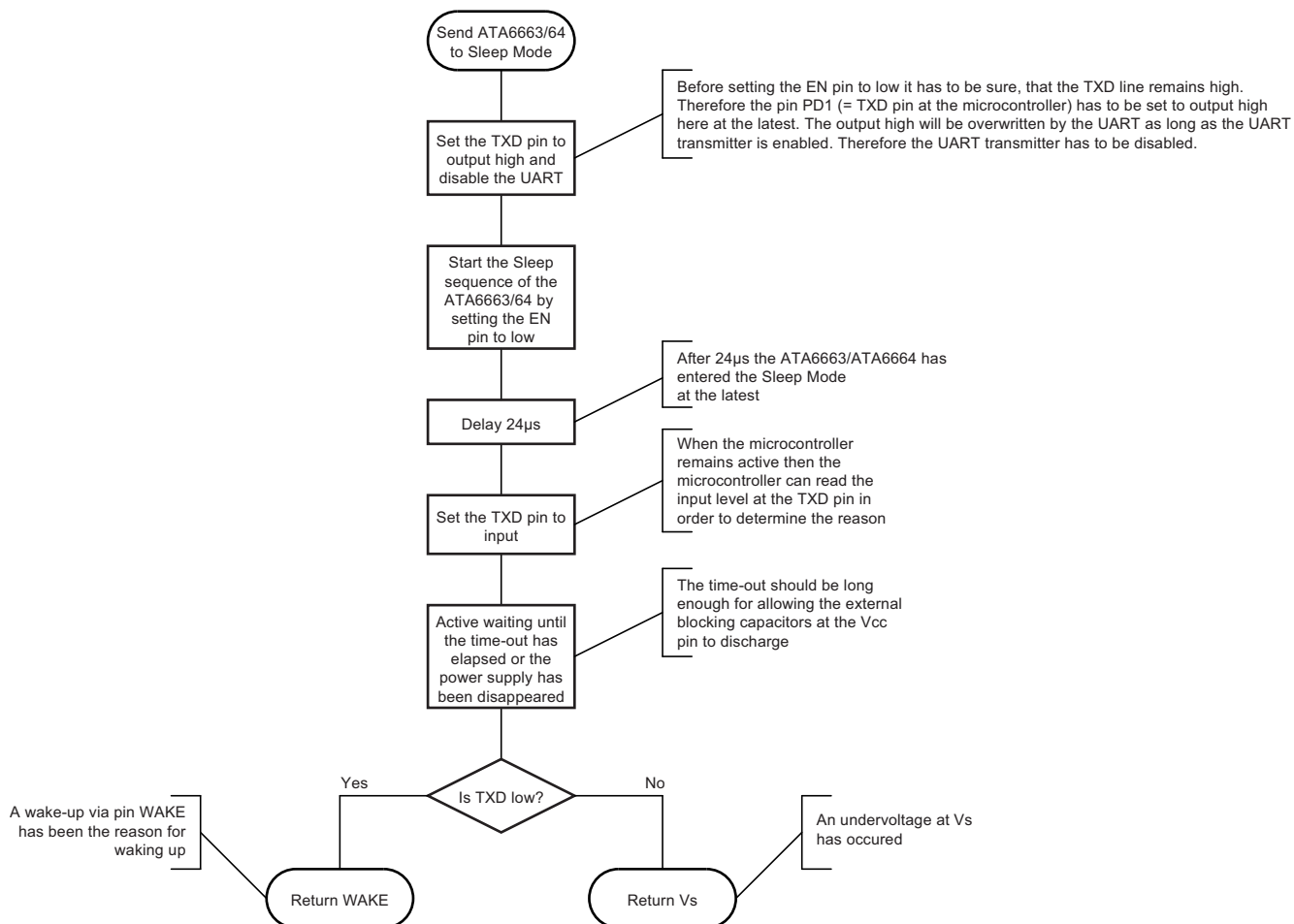
As the variable values are highly application-dependent, it is not possible to state a fixed time-out time that would be valid for all applications in this document.

In order to discharge the blocking capacitors as rapidly as possible, resulting in an earlier undervoltage reset, it is obvious that current consumption should not be reduced by the microcontroller entering a current save mode itself while waiting for the power supply to disappear.

2.3 Program Flow

The following programming flowchart shows a secure way of switching the Atmel® ATA6663/ATA6664 into sleep mode and recognizing the unwanted situations described here, in which the microcontroller stays alive when using the C routine from [Section 1.2 “Basic C Routine for Putting Atmel ATA6663/ATA6664 into Sleep Mode” on page 4.](#)

Figure 2-2. Flowchart



Using this flow, the calling function will be reported as the reason why the Atmel ATA6663/ATA6664 has left the sleep mode and why the microcontroller has remained active. The calling function then has to decide how to proceed.

2.4 Securely Switching to Sleep Mode C Routine

The following C function could be used to securely switch the Atmel® ATA6663/ATA6664 into the sleep mode.

```
unsigned char Switch_ATA6663_64_To_SleepMode (void)
{
    // Before setting the EN pin to low it must be ensured that the TXD line
    // remains high. Consequently, the TXD pin has to be set to output high
    // here at the latest. The output high will be overwritten by the UART as
    // long as the UART transmitter is enabled.
    // Consequently, the UART transmitter must be disabled.
    TXDPIN_high;          // Set the TXD pin at the microcontroller to output high
    TXDPIN_output;        // Set the TXD pin to output in the DDRx register
    UCSRB &= ~(1<<TXEN); // Disable the UART transmitter

    // Now the actual switching of the ATA6663/64 into sleep mode takes place
    ENABLEPIN_low;        // Switch EN pin to low; assuming that the
    // microcontroller pin is already set to output
    // From this point the power supply of the microcontroller will be
    // switched off sooner or later. The microcontroller is waiting for this
    // to happen.

    // Delay 24µs; assuming the device is running on a 1MHz RC oscillator
    // For IAR compiler, the file intrinsics.h is required for
    // using the delay function, and for GCC compiler the file delay.h
    delay_24us; // This function is interpreted according to the compiler used

    // Set the TXD pin to input. The internal pull-ups will then be active
    // because of the former microcontroller-PORT setting (TXDPIN_high)
    TXDPIN_input; // Set the TXD pin @ the microcontroller to input

    // 5ms delay for the power supply to be switched off. The time-out should
    // be adapted to the particular application
    delay_5ms;      // This function is interpreted according to the compiler
                    // used

    // The microcontroller normally never reaches the following statements, as
    // the power supply should have been switched off long before
    // If the TXD pin is high, then either a short undervoltage at Vs was
    // present while switching to sleep mode and the ATA6663/64 was
    // woken up because of this, or
    // If the TXD pin is low, then an immediate and regular wake-up via the
    // WAKE pin (local wake-up) was detected
    if(TXD_INPUT_PORT & (1<<TXDPIN))
        return VS;          // Undervoltage wake-up
    else
        return WAKE;        // Local wake-up
}
```

See [Section 3.1 “Basic Source Code Header-File” on page 14](#) for global parameter definition.

2.5 Wake-up Source Recognition

The device can distinguish between a local wake up request (WAKE pin) and a remote wake-up request (LIN bus). The wake-up source can be read on the TXD pin and the RXD pin in fail-safe mode. If an external pull-up resistor has been connected between the pin TXD and the power supply of the microcontroller, a high level at the TXD pin indicates a remote wake-up request (weak pull-down at pin TXD) and a low level indicates a local wake-up request (strong pull-down at pin TXD). A high level on the RXD pin indicates power-up, and in this application (see [Figure 1-2 on page 3](#)) an undervoltage wake up.

The wake-up request flag (signaled at the RXD pin) as well as the wake-up source flag (signaled at the TXD pin) are reset immediately if the microcontroller sets the EN pin to high.

The microcontroller has to be able to recognize how it was woken up in order to be able to take the right action after the wake-up. For example, after a local wake-up (WAKE pin) the module has to wake up the LIN cluster and wait for commands from master. After a remote wake-up (LIN bus), the module has to prepare to receive data from master. After power-up or undervoltage the module has to initialize itself first and then prepare to receive data from master.

The following basic code routine shows how to carry out the procedures described above.

```
unsigned char Get_Wakeup_Source(void)
{
    if(RXD_INPUT_PORT & (1<<RXDPIN)) // Check the status of the RXD pin
        return VS;    // If high - Undervoltage on the Vs pin or Power up
    else
        if(TXD_INPUT_PORT & (1<<TXDPIN)) // Check the status of the TXD pin
            return LIN;    // If high - Remote wake-up (from LIN bus)
        else
            return WAKE;    // If low - Local wake-up
}
```

See [Section 3.1 “Basic Source Code Header-File” on page 14](#) for global parameter definition.

3. Basic Source Code C-File

The basic source code below shows how to implement the previously described operating modes of ATA666x.

```
#ifdef __IAR_SYSTEMS_ICC__                // If an IAR compiler is used
#include <ioavr.h>
#include <ina90.h>

#elif __GNUC__                            // If GCC compiler is used
#include <avr/io.h>
#endif

#include "ATA666x_Op_Modes.h"
/*****
*
*   Function name:          Main
*
*   Returns:               None
*
*   Parameters:            None
*
*   Purpose:               Contains the main loop of the program
*
*****/
#ifdef __IAR_SYSTEMS_ICC__
__C_task void main(void)

#elif __GNUC__
void main (void)
#endif
{
    unsigned char Wake_Up_Source = POWER_ON;
    l_bool result = FALSE;

    RXDPIN_high;                // Enable pull-up resistor at RXD pin for the
                                // wake-up source recognition. Pin is input per
                                // default
    TXDPIN_high;                // Enable pull-up resistor at TXD pin for the
                                // wake-up source recognition. Pin is input per
                                // default
    Wake_Up_Source = Get_Wakeup_Source(); //Establish the wake-up reason

    // Program initialization
    result = l_sys_init(); // Initialization of the LIN core
    check(result);
    result = l_ifc_init(); // Initialization of the Interface
    check(result);
    ENABLEPIN_high;           // Put the ATA666x into Normal mode
    sbi(EN_DIRECTION_PORT,ENABLEPIN);

    result = l_ifc_connect();  // connect the interface to the LIN cluster
                                // and enable the transmission of headers
                                // and data to the bus

    check(result);
    if (Wake_Up_Source == WAKE)
    {
        l_ifc_wake_up();        // Generate a wake-up on the LIN bus
    }

    while(1)                    // Main loop
}
```

```

{
    // Put the main routine of the LIN node here
    //...
    //...

    Switch_ATA666x_To_SleepMode(); //Put the ATA666x in sleep mode
} //End Main loop
}
/*****
*
*   Function name:   Switch_ATA666x_To_SleepMode
*
*   Returns:        Reason why not in sleep mode
*
*   Parameters:     None
*
*   Purpose:        Safely puts the ATA666x into sleep mode
*
*****/
unsigned char Switch_ATA666x_To_SleepMode (void)
{
    // Before setting the EN pin to low it has to be sure, that the TXD line
    // remains high. Therefore the TXD pin has to be set to output high here
    // at the latest. The output high will be overwritten by the UART as
    // long as the UART transmitter is enabled.
    // Consequently, the UART transmitter must be disabled.
    TXDPIN_high;           // Set the TXD pin @ the microcontroller to output
                           // high
    TXDPIN_output;         // Set the TXD pin to output in the DDRx register
    UCSRB &= ~(1<<TXEN;    // Disable the UART transmitter
    // Now the actual switching of the ATA666x into the Sleep Mode takes
    // place
    ENABLEPIN_low;         // Switch EN pin to low; assuming that the
    // microcontroller pin is already set to output
    // From now on the power supply of the microcontroller will be switched
    // off sooner or later. The microcontroller is waiting for this to
    // happen.

    // Delay 24µs; Assuming the device is running on a 1MHz RC oscillator
    // For IAR compiler including the file intrinsics.h is required for
    // using the delay function and for GCC compiler the delay.h
    delay_24us; // This function is interpreted according to the used
               // compiler

    // Set the TXD pin to input. The internal pull-ups will be active then
    // because of the former microcontroller-PORT setting (TXDPIN_high)
    TXDPIN_input; // Set the TXD pin at the microcontroller to input

    // Delay 5ms for the power supply to be switched off. The time-out should
    // be adapted to the particular application
    delay_5ms; // This function is interpreted according to the used
              // compiler

    // The microcontroller normally never reaches the following
    // statements, as the power supply should have been switched off long ago
    // If the TXD pin is high, then either a short undervoltage at Vs was
    // present while switching to sleep mode and the ATA666x has been woken
    // because of this.

```

```

        // If the TXD pin is low, then an immediate and regular wake-up via the
        // WAKE pin (local wake-up) was detected

        if(TXD_INPUT_PORT & (1<<TXDPIN))
            return VS;    // Undervoltage wake-up
        else
            return WAKE;  // Local wake-up
    }

/*****
 *
 *   Function name:   Get_Wakeup_Source
 *
 *   Returns:        Wake-up Source
 *
 *   Parameters:     None
 *
 *   Purpose:        Gets the wake-up source: Undervoltage, WAKE pin or LIN
 *                   pin
 *
 *****/
unsigned char Get_Wakeup_Source(void)
{
    if(RXD_INPUT_PORT & (1<<RXDPIN))    // Check the status of the RXD pin
        return VS;                    // If high - Undervoltage on the Vs pin or Power up
    else
        if(TXD_INPUT_PORT & (1<<TXDPIN)) // Check the status of the TXD pin
            return LIN;                // If high - Remote wake-up (from LIN bus)
        else
            return WAKE;                // If low - Local wake-up
}

```

3.1 Basic Source Code Header-File

The definitions of the global parameters are made in the corresponding header file as follows. The header file has to be adjusted as appropriate to match the particular hardware used.

```

// ATA666x_Op_Modes.h

#define F_CPU                1000    // Clock in kHz

#ifdef __IAR_SYSTEMS_ICC__    // If IAR compiler is used
#include "intrinsics.h"
#define delay_24us            __delay_cycles(24*F_CPU/1000)
#define delay_5ms             __delay_cycles(5*F_CPU)
#endif

#ifdef __GNU__                // If GCC compiler is used
#include <util/delay.h>
#define delay_24us            _delay_us(24)
#define delay_5ms             _delay_ms(5)
#endif

// Macro definitions
#define sbi(port,bit)         (port |= (1<<bit)) //set bit in port
#define cbi(port,bit)         (port &= ~(1<<bit)) //clear bit in port

```

```

#define ENABLEPIN          0 // Pin0 of PortA
#define EN_PORT            PORTA
#define EN_DIRECTION_PORT  DDRA
#define EN_INPUT_PORT      PINA
#define ENABLEPIN_low      cbi(EN_PORT,ENABLEPIN)
#define ENABLEPIN_high     sbi(EN_PORT,ENABLEPIN)

#define TXDPIN             1 // Pin0 of PortD
#define TXD_PORT           PORTD
#define TXD_DIRECTION_PORT DDRD
#define TXD_INPUT_PORT     PIND
#define TXDPIN_low         cbi(TXD_PORT,TXDPIN)
#define TXDPIN_high        sbi(TXD_PORT,TXDPIN)
#define TXDPIN_output      sbi(TXD_DIRECTION_PORT,TXDPIN)
#define TXDPIN_input       cbi(TXD_DIRECTION_PORT,TXDPIN)

#define RXDPIN             2 // Pin0 von PortA
#define RXD_PORT           PORTA
#define RXD_DIRECTION_PORT DDRA
#define RXD_INPUT_PORT     PINA
#define RXDPIN_low         cbi(RXD_PORT,RXDPIN)
#define RXDPIN_high        sbi(RXD_PORT,RXDPIN)

#define POWER_ON           0
#define VS                 1
#define WAKE               2
#define LIN                3

unsigned char Switch_ATA666x_To_SleepMode (void);
unsigned char Get_Wake-up_Source(void);

```

4. Revision History

Please note that the following page numbers referred to in this section refer to the specific revision mentioned, not to this document.

Revision No.	History
9230B-AUTO-06/15	• Put document in the latest template

