The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

**How to Check the Ordering Part Number**

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

**For More Information**

Please contact your local sales office for additional information about Cypress products and solutions.

**About Cypress**

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

# AN205324

## FR, MB91460, LIN-USART

This application note describes how to use the FR LIN-USART for LIN-Master- and LIN-Slave-Functionality. This note is intended for implementing a low level hardware driver.LIN uses NRZ-8N1L data format in a baud rate range from 9600 Bit/s to 19200 Bit/s. A LIN bus is a single master to multiple slave

## Contents



# 1 Introduction

This application note describes how to use the FR LIN-USART for LIN-Master- and LIN-Slave-Functionality. This note is intended for implementing a low level hardware driver. It neither gives information of the protocol level nor uses LIN-API.

# 2 LIN-Bus

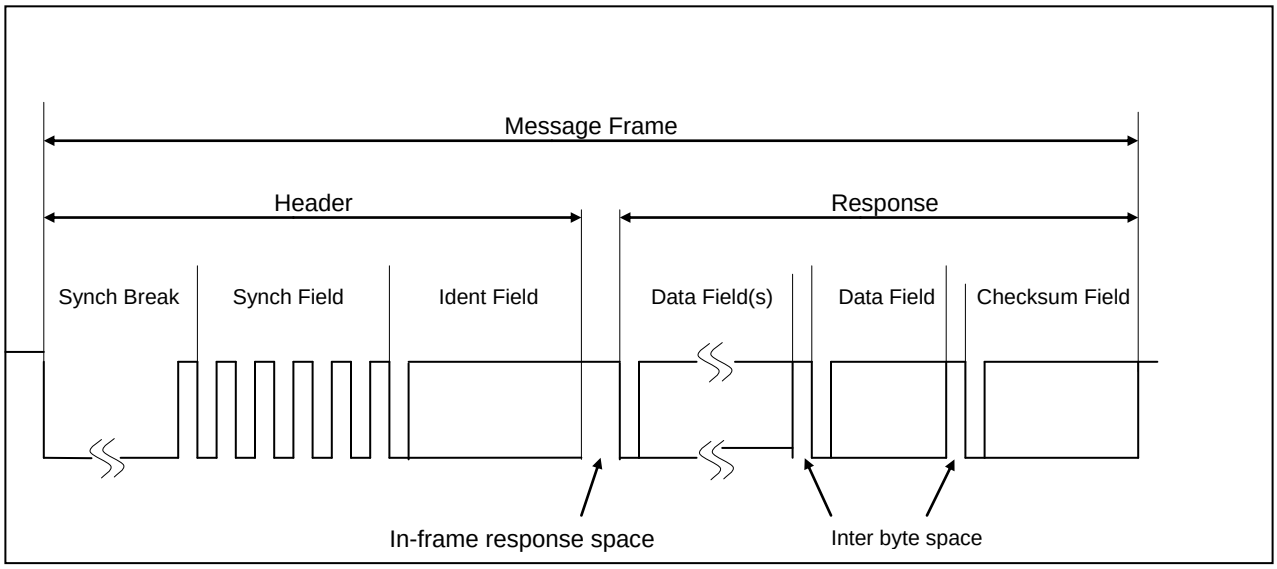
Short Specification

## 2.1 Short LIN Specification

LIN uses NRZ-8N1L data format in a baud rate range from 9600 Bit/s to 19200 Bit/s. A LIN bus is a single master to multiple slave bus.

A LIN message frame consists of a header and a response like in the graphic below:

Figure 1. LIN Message Frame




Except for the Synchronization Break all Fields are simple 8N1L data, this means 1 start bit, 8 data bits (LSB first), no parity, 1 stop bit. This Break is specified as follows:

Figure 2. LIN Synch Break and Synch Field

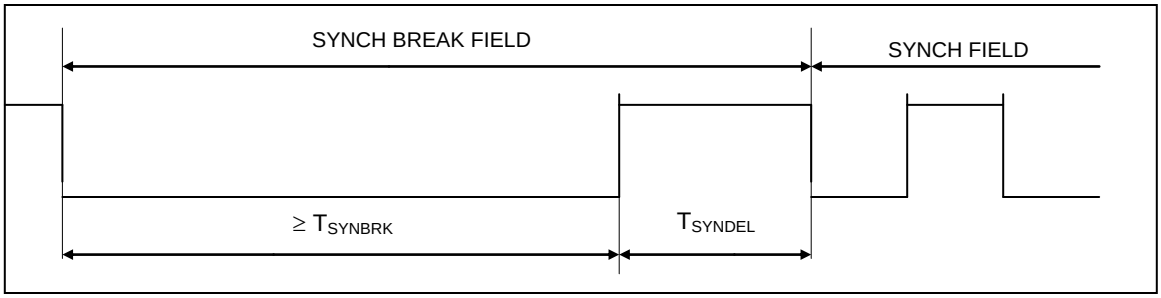
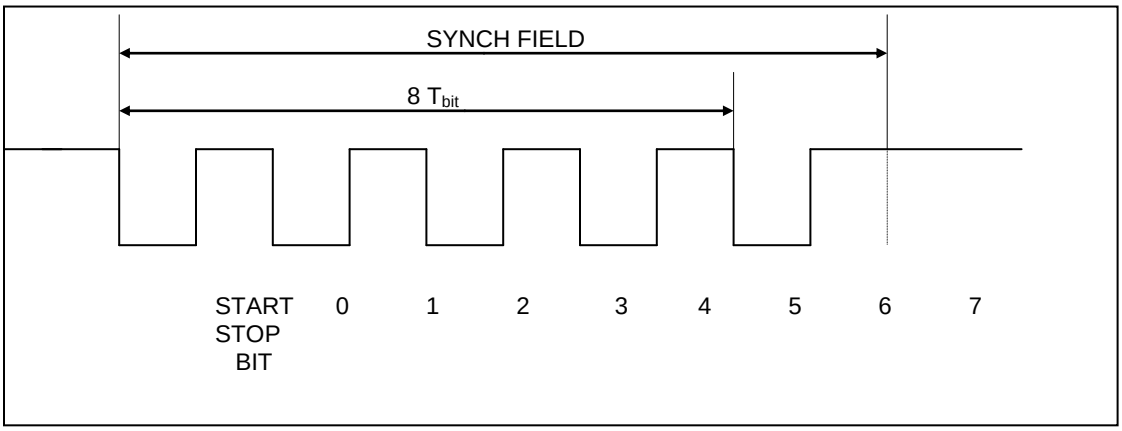


Table 1. Synch Break Timings

| SYNCH BREAK FIELD | LOGICAL | NAME | MIN [$T_{bit}$] | Nom [$T_{bit}$] | MAX [$T_{bit}$] |
|---|---|---|---|---|---|
| SYNCH BREAK LOW PHASE | dominant | TSYNBRK | 13[1] | | - |
| SYNCH BREAK DELIMITER | recessive | TSYNDEL | 1[1] | | - |
| SYNCH BREAK THROSHOLD SLAVE | dominant | TSBRKTS | 11[2] | | |

The Synchronization Field is same as a data filed with the data 0x55 (LSB first). Thus it consists of alternately 5 *dominant and 5 recessive bits:*

Figure 3. Synch Field




All other fields are also simple 8N1L data which contents do not need any special treatment by the hardware.

The checksum is calculated over all data bytes (for LIN 1.3) or over all data bytes and the identifier byte (for LIN 2.1). The calculation is the inverted sum with carry. This means, if the new sum is greater than the old sum by 255 an additional “1” is added. After the last addition, the result is inverted.

The formula for this calculation is:

$$Checksum = 0xFF \oplus \left( \left( \sum_{i=1}^{n} data_i \right) \bmod 0x100 + \left\lfloor \frac{\sum_{i=1}^{n} data_i}{0x100} \right\rfloor \right)$$

Note that in LIN 2.1 $data_i$ also contain the identifier field.

[1] Based on master time base

[2] Based on own slave time base

# 3 LIN-Master

How to Use LIN-USART as Master

## 3.1 LIN Master

The LIN master controls the whole bus. The master starts a LIN message frame with the LIN synchronization break. After that follows the Synchronization Field (same as data field with data 0x55) and the Identifier Field.

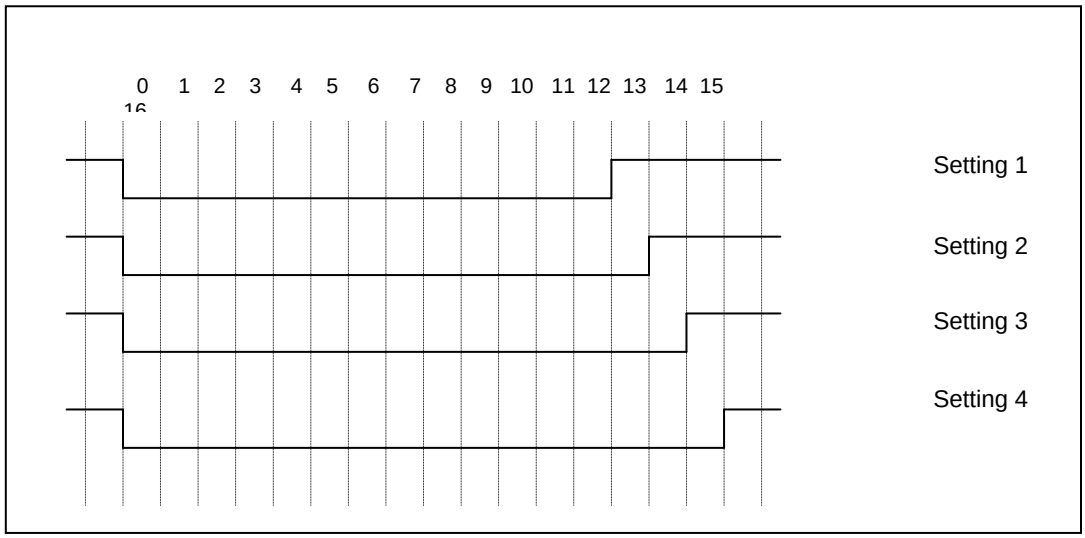
The length of the break can be set with the control bits LBL[1:0] of the Extended Status/Control Register (ESCR) of the LIN-USART.

Table 2. Configuration of Synch Break

| Sr. No. | LBL0 | LBL1 | Length* |
|---|---|---|---|
| 1 | 0 | 0 | 13 $T_{bit}$ |
| 2 | 0 | 1 | 14 $T_{bit}$ |
| 3 | 1 | 0 | 15 $T_{bit}$ |
| 4 | 1 | 1 | 16 $T_{bit}$ |
| * based on master's own time base | | | |

The following figure shows the Synch. Break timing depending on the setting:

Figure 4. Synch Break Timings

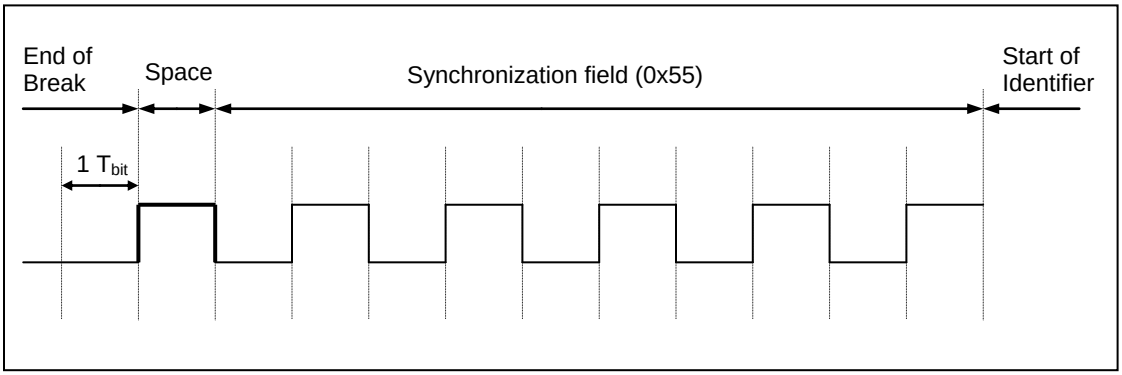



For sending this break LIN-USART has to be in LIN-Mode (mode 3), and the LBR control bit of the Extended Communication Control Register (ECCR) has to be set to 1. At the next internal serial clock the break signal will be generated.

The Synch Field can be sent as a simple 0x55-Byte after the LIN break. To prevent a transmission interrupt, the 0x55 can be written to the TDR register just after writing the "1" to the LBR bit, although the TDRE flag is "0". The internal transmission shifter waits until the LIN break has finished and shifts the TDR value out afterwards. In this case no interrupt is generated after the LIN break and before the start bit.

The following figure illustrates this timing:

Figure 5. Synch Field Timings




Because LIN is a single wire network, the master can read back its own transmissions. Therefore enabling the reception (by setting `RXE` bit of `SCR` register to 1) after a LIN-Break detection (which was set by the master itself) is recommended. The master then receives the synchronization field as a normal `0x55` data and thus is synchronized to the protocol.

Note that after a complete LIN-Message the reception needs to be disabled again, so that the LIN-Break can be detected without getting a framing error.

## 3.2 Example code

The following short example shows how to generate the synch break and field. Note in example USART2 is used.

### 3.2.1 C Code

```
void InitUART2(void)
{
       BGR02 = 832;      // 19200 baud at 16MHz CLKP
       SCR02 = 0x15;     // 8 bit, clear reception errors, Tx enabled,
Rx
       // disabled to avoid Framing error
       SSR02 = 0x00;     // LSB First
       SMR02 = 0xCD;     // Mode 3, Reset Counter, Reset UART, SOT
enabled
       ESCR02 = 0xB0;    // Set Break length to 16 Tbit, Enable Synch.
       // Break Detection interrupt
       PFR20_D0 = 1;     // Enable SIN2 input
       PFR20_D1 = 1;     // Enable SOT2 output
       EPFR20_D1 = 0;    // Enable SOT2 output
}

. . .

void SetBreak(void)
{
       ECCR02_LBR = 1;   // Generate LIN Synch Break
       TDR02 = 0x55;     // Send Synch Field afterwards
}

. . .

__interrupt void LINBreakDetection (void)
{
 ESCR02_LBD = 0;  // Clear LIN Break detection flag
 SCR02_RXE = 1;   // enable Reception for the reception of data
}
```

### 3.2.2 Checksum Calculation

The following code can be used to generate the LIN checksum over the data bytes. Please consider that in LIN 2.1 the identifier byte is also part of the checksum.

```
unsigned int checksum;

void add_checksum(unsigned char adder)
{
       checksum += adder;
}

unsigned char get_checksum(void)
{
       checksum = 0xFF ^ ((checksum & 0x00FF) + (checksum >> 8));
       return ((unsigned char) checksum);
}
```

# 4 LIN-Slave

How to use LIN-USART As Slave

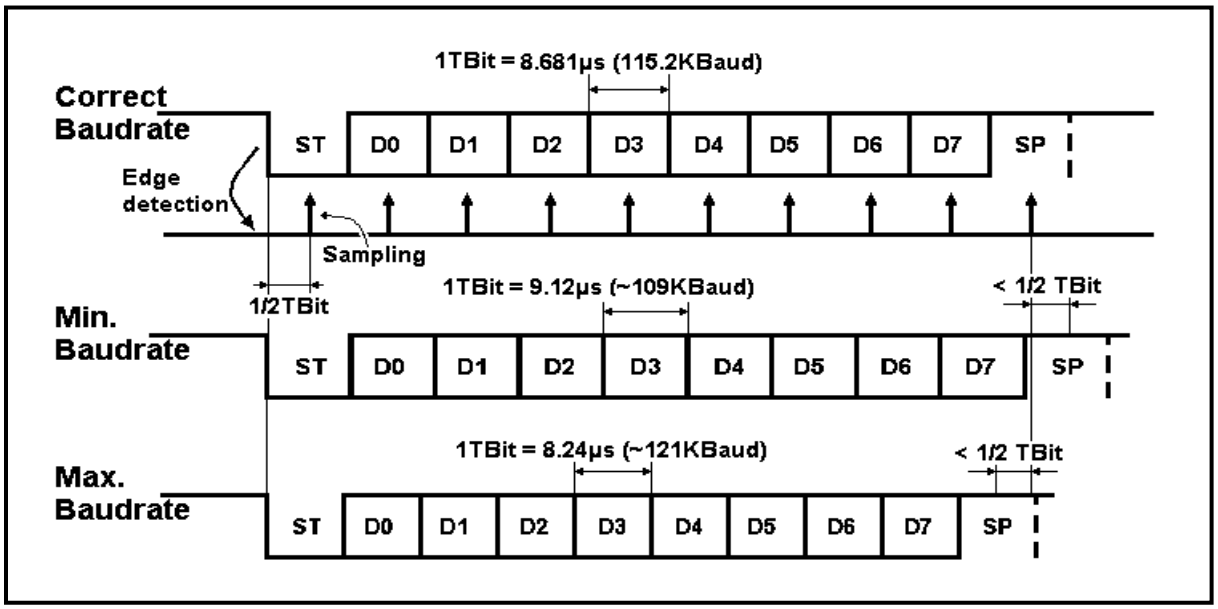
## 4.1 LIN Slave

Any LIN slave is connected to the entire LIN sub bus. To react to a LIN break is stringent. The identifier field then locates a specified slave for responding.

To receive the field such as identifier, data and checksum correctly each slave has to adjust to the LIN master's baud rate. Therefore the slave has to measure the timing of the LIN synchronization field. The result of this measurement is used for the slave baud rate's adjustment.

Note that this measurement is only necessary, if the slave baud rate differs up to ±14% from that of the master baud rate (e.g. when RC oscillator used for slave clock). Because the LIN-USART is synchronizing its internal clock to the falling edge of the start bit, and samples each bit in the middle of the bit time, a theoretical deviation of about ±5% is tolerable (for 8N1 data format). The following illustration shows an example for 115.2k Bits/s.

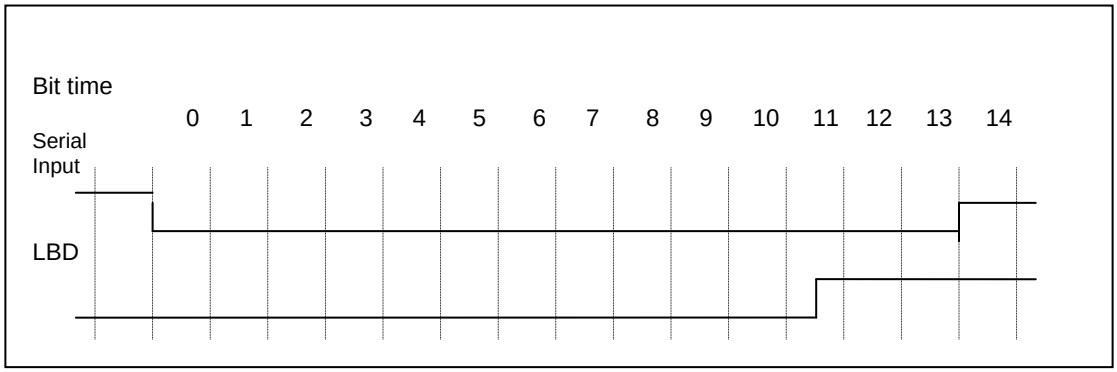
Figure 6. LIN Baud Rate Tolerance



## 4.2 Detecting LIN Synchronization Break

The threshold time of the detection is 11 ½ dominant bits.

Figure 7. Detection of LIN Synch. Break at Slave




If LIN break detection interrupt is enabled (ESCR_LBIE = 1) and normal data reception is disabled (SCR_RXE = 0), a reception interrupt is generated when the ESCR_LBD bit is set by hardware.

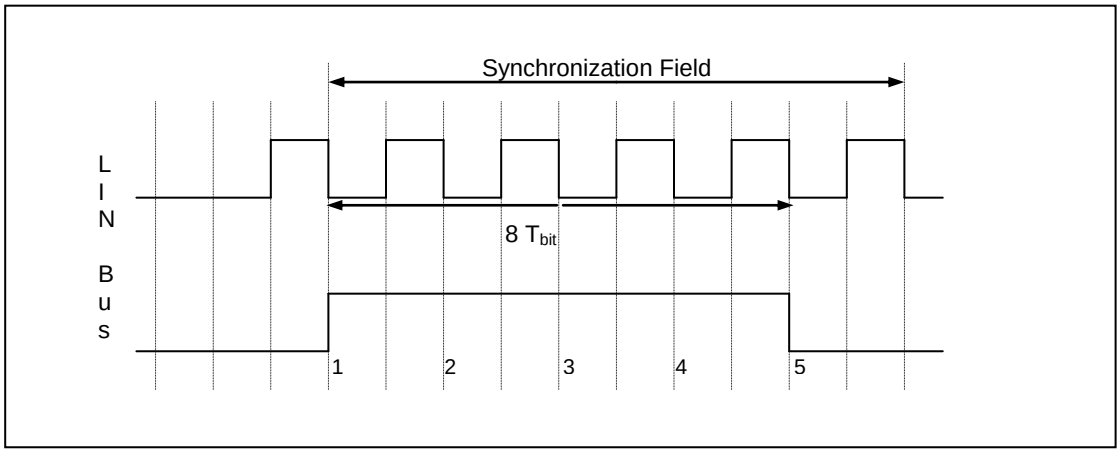
Note the data reception has to be disabled to avoid a framing error (and an additional interrupt) at bit time 9, when a recessive stop bit is expected normally.

The LBD flag has to be cleared by writing “0” to it after a break was detected, even if no interrupts are used.

## 4.3 Baud Rate Measurement

If the LIN-USART has detected a LIN Break it waits for the synchronization field. Within this field an internal signal is generated from the first falling edge to the 5$^{th}$ falling edge. The following graphic illustrates this:

Figure 8. Baud rate measurement




This internal signal is connected to an Input Capture Unit (ICU) of the MCU. Please refer to the hardware manual, which ICU is connectable to which LIN-USART.

The ICU has to be set to “both edge” detection (↑↓). At edge 1 an interrupt is initiated. The user program has to store the actual value of the ICU counter. At edge 5 the second interrupt is generated. The actual value minus the stored divided by 8 is the new baud rate.

This simple algorithm without rounding is accurate enough, so that the calculation result directly can be stored into the baud rate reload counter register. The uncertainty is about 1 LSB anyway, but always within the allowed ±5% tolerance.

Note that the clock divider of the ICU related 16 Bit I/O-Timer has to disabled, i.e. division factor 1, so that the ICU and the USART baud rate generator have the same time base. If other division factors used, the calculation has to be adapted. But be aware that the accuracy decreases with increasing division factor.

If high MCU clocks and low speed LIN baud rates are used, the 16 bit of the timer counter will not be sufficient for counting 8 bits. Either division factors have to be used or the calculation has to regard counter overflows.

**Important Note:** After updating the baud rate, the normal data reception must be enabled again to receive the next LIN message fields! This enabling has to be done within 2 bit times.

## 4.4 Example Code

In the following an example code for handling the LIN message start is given. Note, that the ICU- and USART-Interrupt masks and vectors have to be initialized.

### 4.4.1 C Code

```
void InitUART5(void)
{
   // Initialize UART5 asynchronous LIN mode
   BGR05 = 832;     // 19200 baud at 16MHz CLKP
   SCR05 = 0x01;    // enable transmission
   SMR05 = 0xC5;    // enable SOT5, Reset, LIN mode
   SSR05 = 0x02;    // enable reception interrupt

         PFR19_D4 = 1;           // enable UART: SIN, SOT for async. transfer
         PFR19_D5 = 1;
         EPFR19 = 0x00;          // enable UART

         ESCR05_LBIE = 1;        // enable LIN break detection interrupt
}

void InitICU6(void)
{
   // Initialize Input Capture Unit 6 for LIN-Synch-Field measurement
   PFR14 = 0x20;           // Connect ICU 6 to UART5
   EPFR14 = 0x20;

   ICS67_EG60 = 1;         // Both edge detection
   ICS67_EG61 = 1;
   ICS67_ICP6 = 0;         // Clear possible ICU7-IRQ
   ICU_State = 0;          // State flag
}



void main(void)
{
         __EI();                 // enable interrupts //
       __set_il(31);           // allow all levels //
       InitIrqLevels();        // init interrupts //

       PORTEN = 0x3;           // enable I/O Ports //
       InitUart5();
   InitICU6();

   Rx_Error = 0;
   LIN_State = 1;
   while(1)
   {
         HWWD_CL = 0;
   }
}
```

```
__interrupt void RxIRQHandler(void)
{
       HWWD = 0x10;                          // reset hardware watchdog
 if (ESCR05_LBD)                    // LIN Break Detection?
 {
       ESCR05_LBD = 0;              // Clear flag

       if (LIN_State == 1)
       {
              SCR05_RXE = 1;        // Enable reception
              LIN_State = 2;        // ready to receive synch field
              LIN_Checksum = 0;
              LIN_Count = 0;

              ICS45_ICP4 = 0;       // Clear possible ICU2-IRQ
              ICS45_ICE4 = 1;       // Enable ICU2-Interrupt
       }

    else
    {
      Rx_Error = 1; // Unexpected reception of break
    }
   }
else if (LIN_State == 3)       // Header read?
    {
              LIN_Header = Rx_Data;

       if (LIN_Header == SLAVESEND)
       {
              TDR05 = LIN_Data[LIN_Count];// Send LIN Data
              LIN_Count++;
       }

       LIN_State = 4;
    }
    else if (LIN_State == 4)  // LIN Data read / write
    {
       if (LIN_Header == MASTERSEND)// Master sent data?
       {
              LIN_Checksum = LIN_Checksum + Rx_Data;
              LIN_Data[LIN_Count] = Rx_Data;
              LIN_Count++;
              // End of message reached?
              if (LIN_Count == DATALENGTH
              {
                     LIN_Count = 0;
                     LIN_State = 5;
                     LIN_Checksum = LIN_Checksum ^ 0xFF;
              }
       }

       else
       {
         LIN_Checksum = LIN_Checksum + IN_Data[LIN_Count];
        TDR05 = LIN_Data[LIN_Count];// Send next LIN Data
       LIN_Count++;
```

```
// End of message reached?
if (LIN_Count == DATALENGTH)
   {
    LIN_Count = 0;
    LIN_State = 5;
    LIN_Checksum = LIN_Checksum ^ 0xFF;
   }
 }
}

else if (LIN_State == 5)// LIN Checksum read / write
{
 if (LIN_Header == MASTERSEND)// Master sent data?
 {
   if (Rx_Data != LIN_Checksum)
   {
     Rx_Error = 4;
   }
 }
 else
 {
   TDR05 = LIN_Checksum; // Send Checksum
 }

    SCR05_RXE = 0; // disable reception wait for LIN break
    LIN_State = 1; //  (new message)
   }
 }

 else          // Not recognized interrupt cause
 {
   Rx_Error = 5;
   SSR05_RIE = 0;    // disable reception interrupt
 }
}
__interrupt void ICU6_IRQHandler(void)
{
 if (ICU_State == 0)        // Rising edge detected?
 {
   ICU_Value1 = IPCP6;
   ICU_State = 1;
 }
 else        // Falling edge (last edge) detected!
 {
   ICU_Value2 = IPCP6;
   ICU_State = 0;
   BGR5 = (ICU_Value2 - ICU_Value1) / 8;
 }
 ICS67_ICP6 = 0;           // Clear ICU6-IRQ
}
```

### 4.4.2 Assembly Code

```
#set      RXE 0             ; Bit positions
#set      RIE 0
#set      ICUSj  4
#set      EGj0   2
#set      EGj1   3
#set      ICPj   7
#set      LBD 6

InitUARTn:   MOVW   BRGn,#0682          ; 1666 => 9600 Baud @ 16 MHz
          MOV SMRn,#CE            ; Reset, Lin mode
          CLRB   SCRn:RXE         ; No reception
          STEB   SSRn:RIE         ;  ... but reception interrupt
          RETP

InitICUj:SETB   I:ICEij:ICUSj     ; Connect ICUj to UARTn
          SETB   I:ICSij:EGj0     ; Both edge detection
          SETB   I:ICSij:EGj1
          CLRB   I:ICSij:ICPj     ; Clear possible ICUj-IRQ
          RETP

. . .

Main:        CALLP  \InitIrqLevels      ; initialize IRQ vectors and levels
          MOV ILM,#07             ; allow all levels
          OR  CCR,#40             ; globally enable interrupts

          MOV A,#0
       MOV   LIN_State,A          ; (somewhere in RAM area)

          CALLP  \InitUARTn
          CALLP  \InitICUj

. . .

; interrupt handler for UARTn reception
irqUARTn:BBC ESCRn:LBD,NoBreak

          CLRB   ICSij:ICPj             ; Clear possible ICUj-IRQ
          SETB   ICSij:ICEj             ; enable ICUj-Interrupt
          CLRB   ESCRn:LBD        ; Clear detection flag
          MOV A,#1
          MOV LIN_State,A
          RETI

NoBreak: . . .                    ; Handle other reception IRQ cause


; interrupt handler for ICUj
irqICUj: MOV A,LIN_State
       MOVN  A,#1
       CMPW  A
          BNZ FallEdge

          MOVW   A,IPCPj          ; rising edge
          MOVW   ICU_Value1,A     ; (somewhere in RAM area)
          INC LIN_State
          BRA irqEnd
```

```
FallEdge: PUSHW  RW4              ; falling edge

        MOVW  A,IPCPj             ; calculate and set new baud rate
        MOVW  RW4,ICU_Value1      ; (ICU_new - ICU_old) -> A
        SUBW  A,RW4
        LSRW  A             ; (A / 8) -> A
        LSRW  A
        LSRW  A
        MOVW  BGRn,A        ; A -> BGRn

        POPW  RW4
        INC   LIN_State

IrqEnd: CLRB  ICSij:ICPj          ; clear ICUj-IRQ
        RETI
```

### 4.5 Checksum Calculation

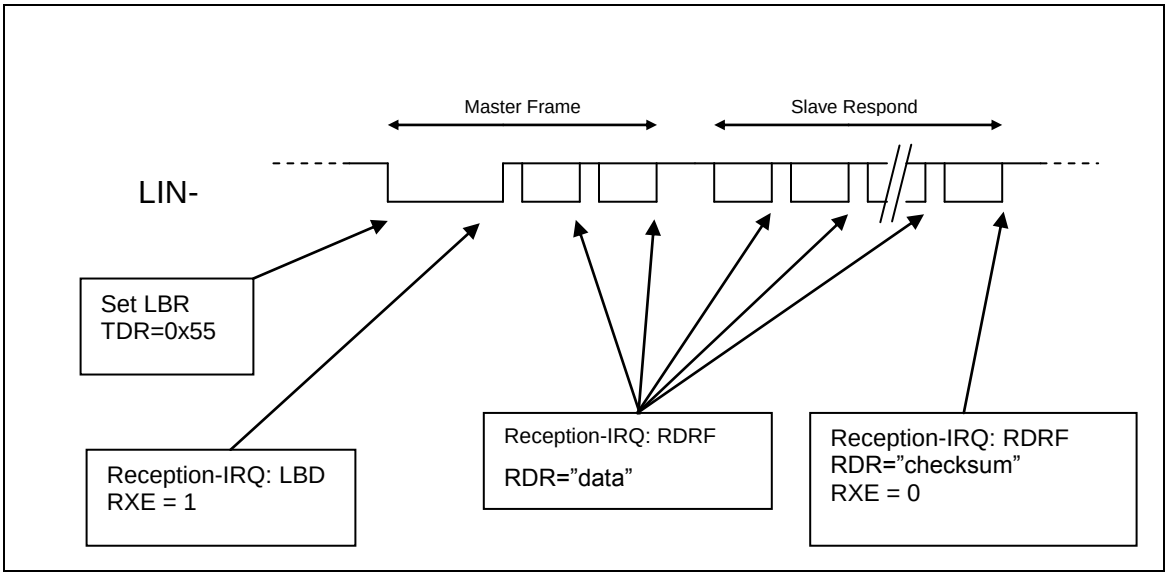
The checksum calculation is the same as for the LIN master (3.2.1).

## 5 Overview about Interrupt Load

Interrupts during LIN frame

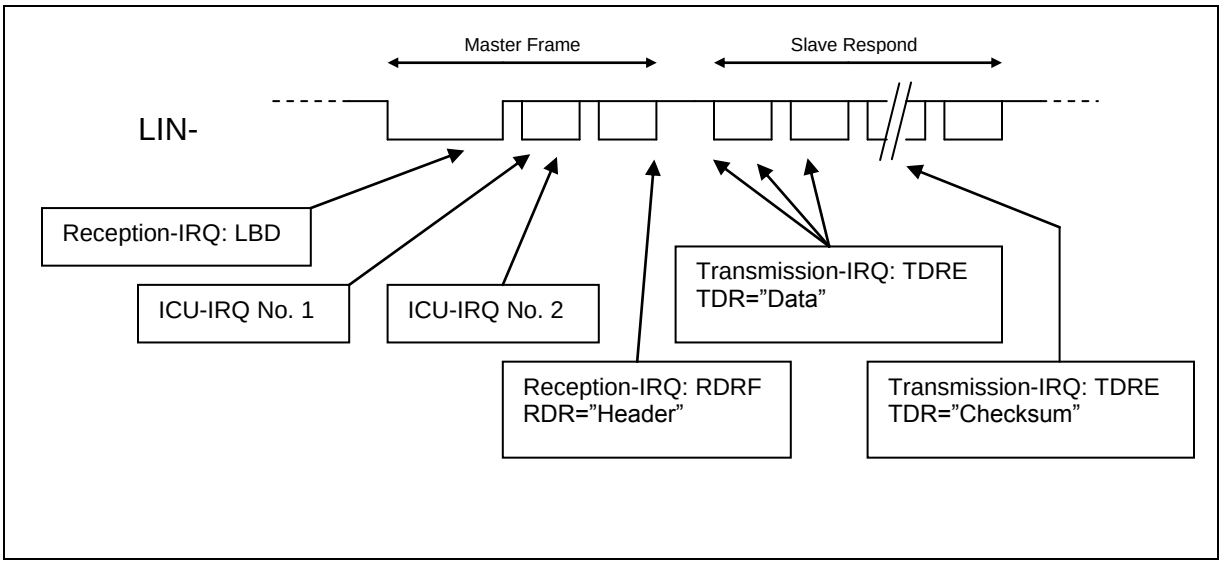
### 5.1 LIN-USART as LIN Master

The following illustration gives an overview about the interrupt load when LIN-USART is bus master:



### 5.2 LIN-USART as LIN Slave

The following illustration gives an overview about the interrupt load when LIN-USART is bus slave:




## 6 Error Handling

How to Handle Hardware Errors

### 6.1 Bus broken

If the LIN-Bus is broken, the master gets no response from the slaves. In this case the application will indicate a no-slave-response error.

### 6.2 Bus shortened to VCC

If the LIN-Bus is shortened to the supply voltage, the bus is every time recessive. The application can indicate this error by the missing master-synch-break-read back or also no-slave response error.

### 6.3 Bus shortened to GND

If the LIN-Bus is shortened to the ground, the bus is every time dominant. The LIN-Master would read-back the LIN-Synch-Break, but no read-back the 0x55 byte of the Synch Field. The application can indicate this error by wrong read-back* or no-slave-response.

**Important note:** If the serial input of the LIN-USART is connected to ground, the internal LIN-Break-detection circuit will always set the LBD (LIN Break Detection) Flag after each 11 bit times. Be careful using interrupts. Be sure to clear *all* reception interrupt causes in the interrupt handler routine.

* since LIN Specification V 2.0 master read back is not enjoined anymore.

# 7 Notes on using USART Programmable Clear (UPCL)

How to use UPCL

## 7.1 UPCL Handling

Generally **don't set** the **UPCL**-Bit ("Software Reset") of the Serial Mode Register (SMR) when **TXE**-Bit of the Serial Control Register (SCR) is "**1**".

# 8 Additional Information

Information about CYPRESS Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software examples related to this application note is:

*91460_uart_lin_master*

*91460_uart_lin_slave*

It can be found on the following Internet page:

http://www.cypress.com/cypress-mcu-product-softwareexamples

## Document History

Document Title: AN205324 - FR, MB91460, LIN-USART

Document Number:002-05324

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | - | NOFL | 07/08/2008 | Initial Version, MPi |
| | | | 07/13/2008 | Update Chapters 3 to 9 HPi |
| | | | 09/23/2008 | MSt, typos removed |
| *A | 5071244 | NOFL | 02/09/2016 | Converted Spansion Application Note “MCU-AN-300073-E-V12” to Cypress format |
| *B | 5844362 | AESATP12 | 08/04/2017 | Updated logo and copyright. |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| ARM® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

## Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709