AN205498

F²MC 16FX Family, USART

This application note describes the functionality of the USART and their operation modes

Contents

1 Introduction....1
1.1 Key Features....1
2 The USART with LIN functionality....2
2.1 Block Diagram....2
2.2 Basic Functionality....3
2.3 Registers....3
2.4 Error Conditions....8
2.5 Interface to the BUS....12
2.6 DMA Transmission with USART....13
3 USART Examples....15
3.1 Asynchronous Mode (0) without Interrupts....15
3.2 Asynchronous Mode (0) with Interrupts....16
3.3 Synchronous Mode (SPI) Master without Interrupts....17
3.4 DMA block transmission....21
4 APPENDIX A....23
4.1 Related Documents....23
5 Additional Information....23
6 Document History....24
Worldwide Sales and Design Support....25
Products....25
PSoC® Solutions....25
Cypress Developer Community....25
Technical Support....25

1 Introduction

This application note describes the functionality of the USART, their operation modes and gives some examples.

1.1 Key Features

- Full Duplex
- NRZ/RZ for serial Data In- and Output
- NRZ/RZ for serial Clock In- and Output
- Asynchronous and synchronous Mode
- 7-8 Data Bits and 1-2 Stop Bits for asynchronous mode, even or odd Parity selectable
- Dedicated Reload Counter as Clock Divider for Baud Rate (610 Bits/s up to 4 MBits/s synchronous at 20 MHz Peripheral Clock)
- Reload Counter can be fed with external clock
- Synchronous master or slave capable
- 4 SPI Clock Modes
- Framing, Overrun, and Parity Error detectable
- LIN Synch Break Detection and Generation (13, 14, 15, 16 Bit Times selectable)
- LIN Synch Field Signal can be fed to Input Capture Unit for Time Measurement
- Start and Stop Bits selectable in synchronous Mode
- Continuous Serial Clock Output selectable in synchronous Mode
- Asynchronous Master-Slave Communication (Address and Data Bit selectable)

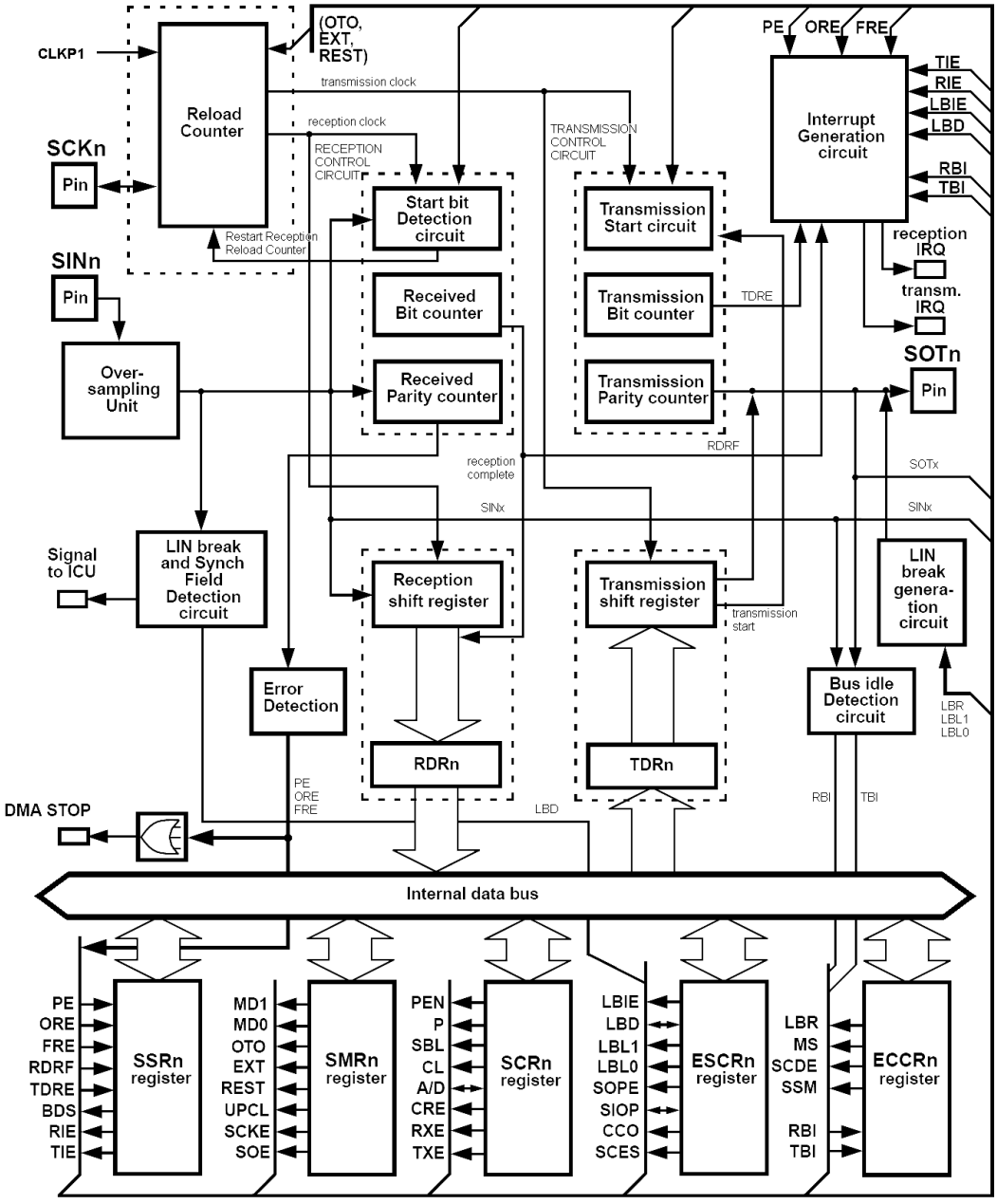
2 The USART with LIN functionality

The Basic Functionality of the USART with LIN Functionality

2.1 Block Diagram

Figure 1 shows the internal block diagram of the USART.

Figure 1. USART Block Diagram



2.2 Basic Functionality

The USART has four different operation modes, which are selectable via the MD [1:0] bits in the Serial Mode Register (SMR) 2.3.2.

Table 1. USART Operation Modes

MD1	MD0	Mode	Description
0	0	0	Asynchronous (normal mode)
0	1	1	Asynchronous (Master/Slave Mode)
1	0	2	Synchronous Mode
1	1	3	Asynchronous (LIN Mode)

Please change the mode only when USART reception and transmission is off `(SCR: RXE = 0, SCR: TXE = 0)`. Furthermore it is recommended to reset the USART with the SMR: UPCL bit, after operation mode has changed.

Example:

```
   . . .
SMR0 = 0x83; // Set USART to Mode 2 (synchronous operation), serial output &
       // serial clock output enabled
SMR0 = 0x87; // Reset USART (UPCL bit#2 is auto-cleared)
   . . .
```

2.3 Registers

Please note that any changes in the settings of the USART should be done while reception as well as transmission is disabled `(RXE = 0, TXE = 0)`. Otherwise the result of ongoing reception / transmission might be incorrect and the USART might not be initialized correctly.

2.3.1 Serial Control Register (SCR)

Table 2. SCR

Bit No.	Name	Explanation	Value	Operation
7	PEN	Parity Enable*[1]	0	Parity disabled
			1	Parity enabled
6	P	Parity Even/Odd Selection*[1]	0	Even Parity enabled
			1	Odd Parity enabled
5	SBL	Stop Bit Length*[2]	0	1 Stop bit selected
			1	2 Stop bits selected
4	CL	Character Length*[3]	0	7 Bits
			1	8 Bits
3	AD	Address/Data Bit*[4]	0	Data Bit
			1	Address Bit
2	CRE	Clear Reception Errors	0	Write: No effect
			1	Write: Errors cleared, Reception is reset
1	RXE	Reception Enable	0	Reception disabled
			1	Reception enabled
0	TXE	Transmission Enable	0	Transmission disabled
			1	Transmission enabled

Note: When enabling reception, please also enable the corresponding input pin with its Port Input Enable Register (PIER) bit.

*1 this function is only available in Mode 0 and 2 if SSM = 1.

*2 this function is only available in Mode 0, 1 and 2 if SSM = 1.

*3 this function is only available in Mode 0 and 1.

*4 this function is only available in Mode 1.

2.3.2 Serial Mode Register (SMR)

Table 3. SMR

Bit No.	Name	Explanation	Value	Operation
7,6	MD1,MD0	Mode Bits	0, 0	Asynchronous Normal Mode
			0, 1	Asynchronous Multiprocessor Mode
			1, 0	Synchronous Mode
			1, 1	Asynchronous LIN-Mode
5	OTO	One-to-One External Clock*5	0	Use external Clock with Baud Rate Generator
			1	Use external Clock as is
4	EXT	External Clock Source	0	Use internal Clock with Baud Rate Generator
			1	Use external Clock Source
3	REST	Restart Baud Rate Generator*6	0	Write: No effect
			1	Restart Baud Rate Generator
2	UPCL	USART Programmable Clear*6*7	0	Write: No effect
			1	Write: Reset USART
1	SCKE	Serial Clock Output Enable	0	SCK Pin: Port Function or Clock Input
			1	SCK Pin: Clock Output
0	SOE	Serial Output Enable	0	SOT Pin: Port Function
			1	SOT Pin: Data Output

Note: If synchronous slave mode is selected, SCK pin input must be enabled using corresponding Port Input Enable Register (PIER).

*5 this function is used, if USART should act as Serial Synchronous Slave Device (Mode 2).

*6 these bits are auto-cleared when set

*7 Reset the USART, only if reception and transmission is disabled (RXE = 0, TXE = 0).

2.3.3 Serial Status register (SSR)

Table 4. SSR

Bit No.	Name	Explanation	Value	Operation
7	PE	Parity Error*[8]	0	No Parity Error
			1	Parity Error Detected
6	ORE	Overrun Error	0	No Overrun Error
			1	Overrun Error occurred (New Reception, if RDRF = 1)
5	FRE	Framing Error*[9]	0	No Framing Error
			1	Framing Error Detected (No Stop Bit received)
4	RDRF	Reception Data Register Full	0	No Reception
			1	Reception Data Register is full
3	TDRE	Transmission Data Register Empty	0	Transmission Data Register is full
			1	Transmission Data Register is empty (Ready for Transmission)
2	BDS	Bit Direction*[10]	0	LSB first
			1	MSB first
1	RIE	Reception Interrupt Enable	0	Interrupt disabled
			1	Interrupt enabled
0	TIE	Transmission Interrupt Enable	0	Interrupt enabled
			1	Interrupt disabled

*[8] this flag is only available in Mode 0, 1, and 2 if SSM = 1.

*[9] this flag is only available in Mode 0, 1, and 3.

*[10] This function is not available in Mode 3 (LIN).

2.3.4 Reception Data Register (RDR)

This 8-Bit Register contains the received data. This is indicated by the RDRF flag of the Serial Status Register. The RDRF flag also generates the reception interrupt, if enabled.

2.3.5 Transmission Data Register (TDR)

The 8-Bit Register contains the data to be sent. If it is empty (default) the TDRE flag is “1”. Once the data is written to TDR the TDRE flag is cleared (to “0”). Then the data gets shifted to the transmission shift register. Subsequently on the next transmission clock the start bit is transmitted on the bus. After the start bit is transmitted on the subsequent transmission clock the TDRE bit is set to “1” again.

The TDRE flag also generates the transmission interrupt, if enabled. Note, that it is “1” after Reset.

2.3.6 Extended Status/Control Register (ESCR)

Table 5. ESCR

Bit No.	Name	Explanation	Value	Operation
7	LBIE	LIN Break Detection Interrupt enable*11	0	Interrupt disabled
			1	Interrupt enabled
6	LBD	LIN Break Detection Flag/Clear*11	0	Write: Clear LIN Break Detection Interrupt
			1	LIN Break detected
5, 4	LBL1, LBL0	LIN Break Generation Length*11	0, 0	13 Bit Times
			0, 1	14 Bit Times
			1, 0	15 Bit Times
			1, 1	16 Bit Times
3	SOPE	Serial Output Pin Enable	0	SOT Pin is Serial Out
			1	SOT Pin is state of SIOP
2	SIOP	Serial Input/output Pin	0	Write: Force SOT to “0”, if SOPE=1, Read: State of SIN
			1	Write: Force SOT to “1”, if SOPE=1, Read: State of SIN
1	CCO	Continuous Clock Output*12	0	Continuous Clock disabled
			1	Continuous Clock enabled
0	SCES	Serial Clock Edge Selection (inverted CPOL) *13	0	Clock with Mark Level “1”
			1	Clock with Mark Level “0”

*11 this bit is only available in Mode 3 (LIN).

*12 this bit is only available in Mode 2. Only reasonable with SSM=1.

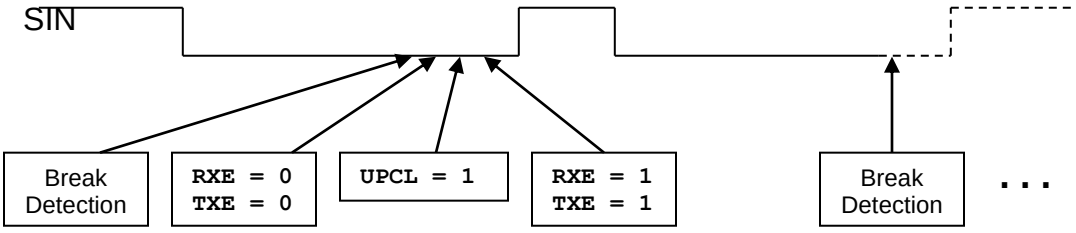
*13 this bit is only available in Mode 2.

LIN Break Detection Consideration

There may an application that requires LIN Break Detection without Synch. Field Detection. In such case, after the LIN break has been detected perform the following steps:

- Disable the transmission and reception by clearing TXE and RXE bits of SCR register respectively.
- Reset the LIN-USART by setting the UPCL bit of SMR register
- And then enable the transmission and reception again by setting TXE and RXE bits of SCR register respectively

This ensures the correct detection of a possible following break frame.



2.3.7 Extended Communication Control register (ECCR)

Table 6. ECCR

Bit No.	Name	Explanation	Value	Operation
7	INV	Invert Serial Data*14	0	Serial Data is not inverted (NRZ format)
			1	Serial Data is inverted (RZ format)
6	LBR	Generate LIN Break*15	0	Ignored
			1	Write: Set LIN Break
5	MS	Synchronous Master/Slave Select*16	0	Synchronous Master Mode
			1	Synchronous Slave Mode
4	SCDE	Synchronous Serial Clock Delay (inverted CPHA) *16	0	No Clock Delay
			1	Clock Delay of half Bit Time
3	SSM	Synchronous Start/Stop Bit Mode*16	0	No start/stop bits in synchronous mode 2
			1	Enable start/stop bits in synchronous mode 2
2	BIE	Bus idle interrupt enable	0	Disable bus idle interrupt
			1	Enable bus idle interrupt
1	RBI	Reception Bus Idle*17	0	Activity on SIN, Reception ongoing
			1	Bus Idle, no Reception ongoing
0	TBI	Transmission Bus Idle*18	0	Activity on SOT, Transmission ongoing
			1	Bus Idle, no Transmission

*14 please note, that setting INV bit causes 0-level at SOT for the first transfer at serial synchronous slave mode, even if TDR was not written to.

**15 this bit is only available in Mode 3 (LIN).

**16 this bit is only available in Mode 2.

*17this flag bit cannot be used in synchronous slave Mode 2.

*18this flag bit cannot be used in synchronous slave Mode 2 (when MS=1).

2.3.8 Baud Rate Generator Register (BGR)

This 15-Bit register contains the divider for the baud rate. The value “v” for the divider can be calculated as:

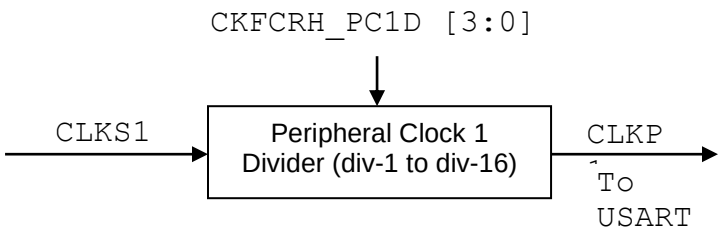
v = [Φ / b] - 1,

Where, Φ is the peripheral clock CLKP1,

b the baud rate

And [] Gaussian brackets (mathematical rounding function).

Figure 2. USART Clock



Minimum and Maximum Ratings

For synchronous operation the minimum divider is 5 to ensure correct internal signal processing.

Because the USART has an internal 5-times over-sampling unit in mode 0, 1 and 3, it is recommended to use also a divider not less than 5 for asynchronous communication.

The maximum divider in all operation modes is 32768. If this division factor is insufficient, the Peripheral Clock has to be divided then.

Table 7. Baud Rate corresponding to CLKP1

Peripheral Clock CLKP1	Minimum Baud Rate (div = 32768)	Maximum Baud Rate (div = 5)
16 MHz	488 Bits/s	3.2 MBits/s
20 MHz	610 Bits/s	4 MBits/s
24 MHz	732 Bits/s	4.8 MBits/s
25 MHz	763 Bits/s	5 MBits/s
48 MHz	1465 Bits/s	9.6 MBits/s

For the relationship between the baud rate and reload values of baud rate generator at different peripheral clock CLKP1 frequencies please refer the hardware manual.

It should also be noted that there would be a deviation between the desired baud rate and the actual baud rate depending upon the CLKP1 clock frequency and reload value. This deviation would be further affected if the CLKP1 is fed from the CLKMOD (clock modulator). Hence the resultant deviation would be dependent on the phase skew of clock modulator which indeed is dependent on configuration such as resolution and modulation degrees at a given PLL frequency (CLKPLL).

2.4 Error Conditions

2.4.1 Clearing Reception Errors and De-synchronization

CRE bit of SCR register resets reception state machine and next falling edge at SINn input starts reception of new byte. This behavior is shown in Figure 3. Therefore either set CRE bit immediately (within half bit time) after receiving errors to prevent data stream de-synchronization as shown in Figure 4 or wait an application dependent time after receiving errors and set CRE, when SINn input is idle.

Figure 3. CRE Bit Timing

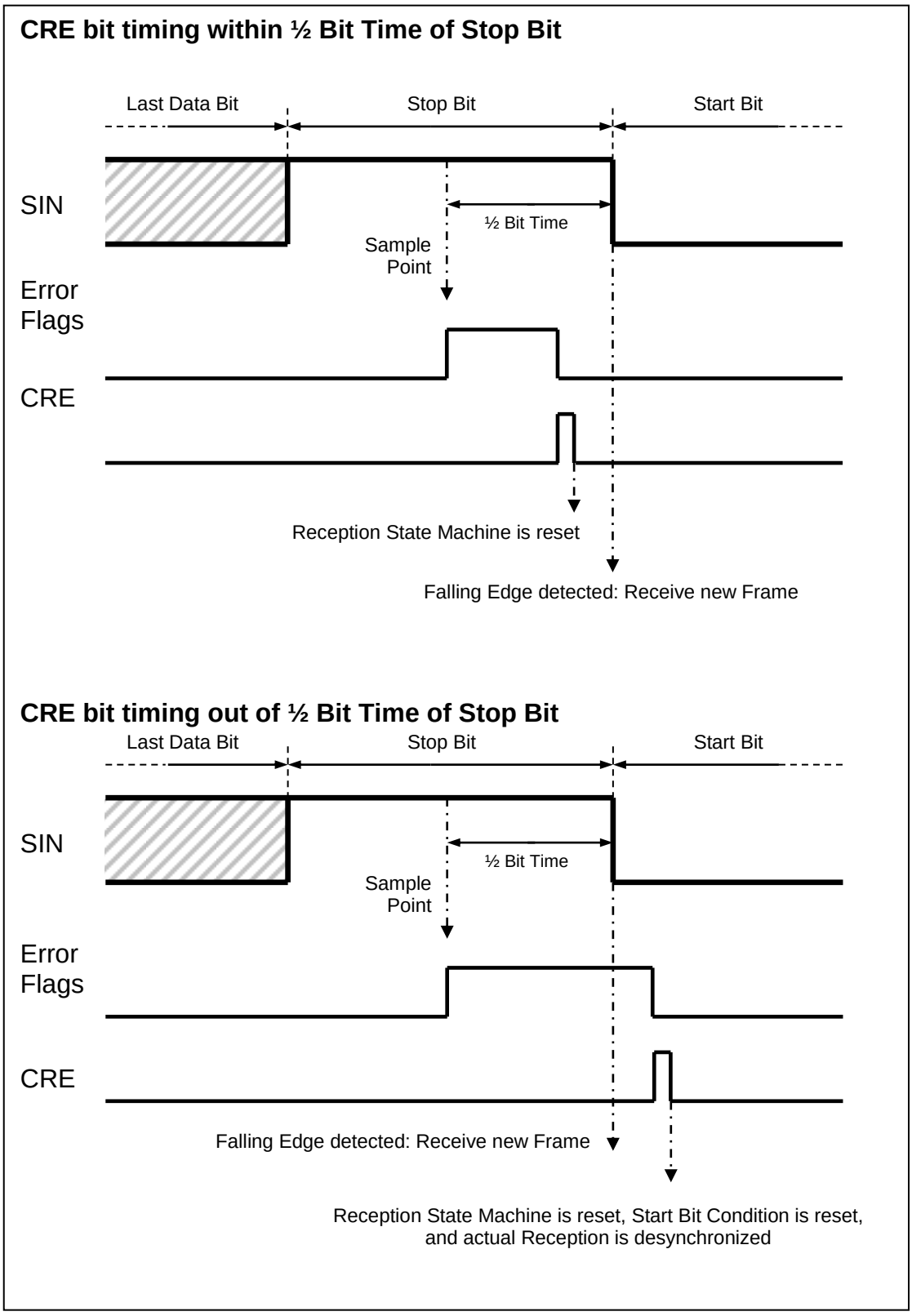
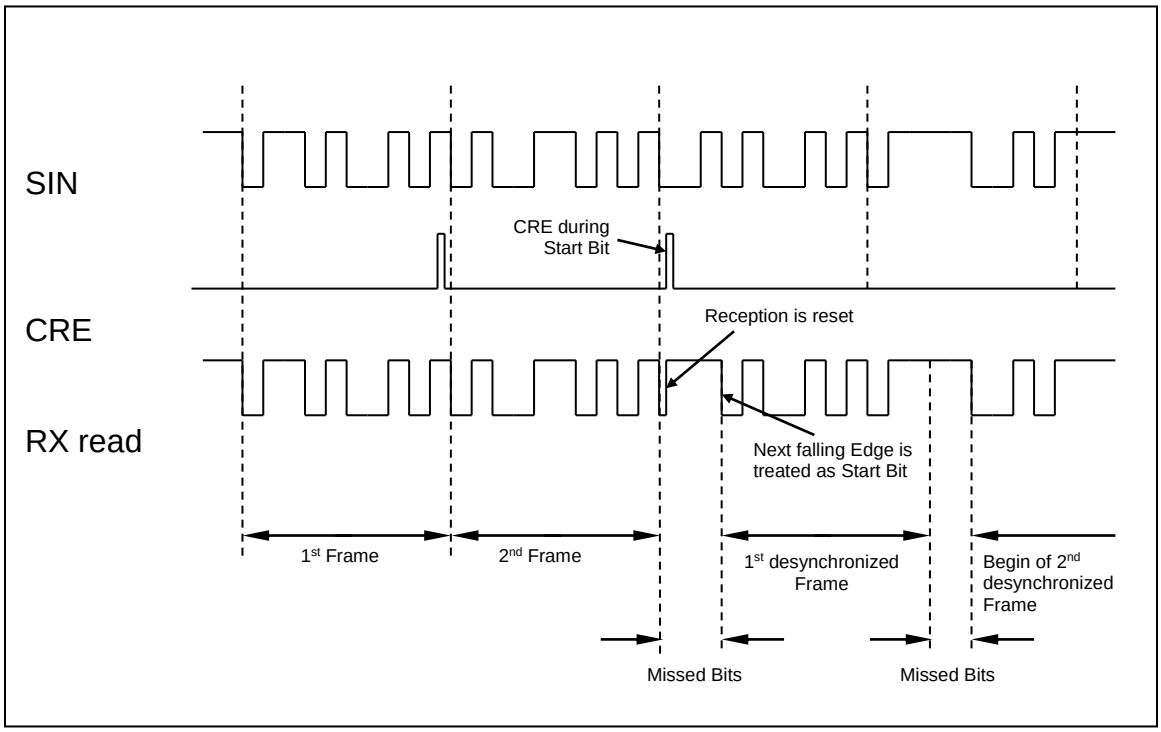


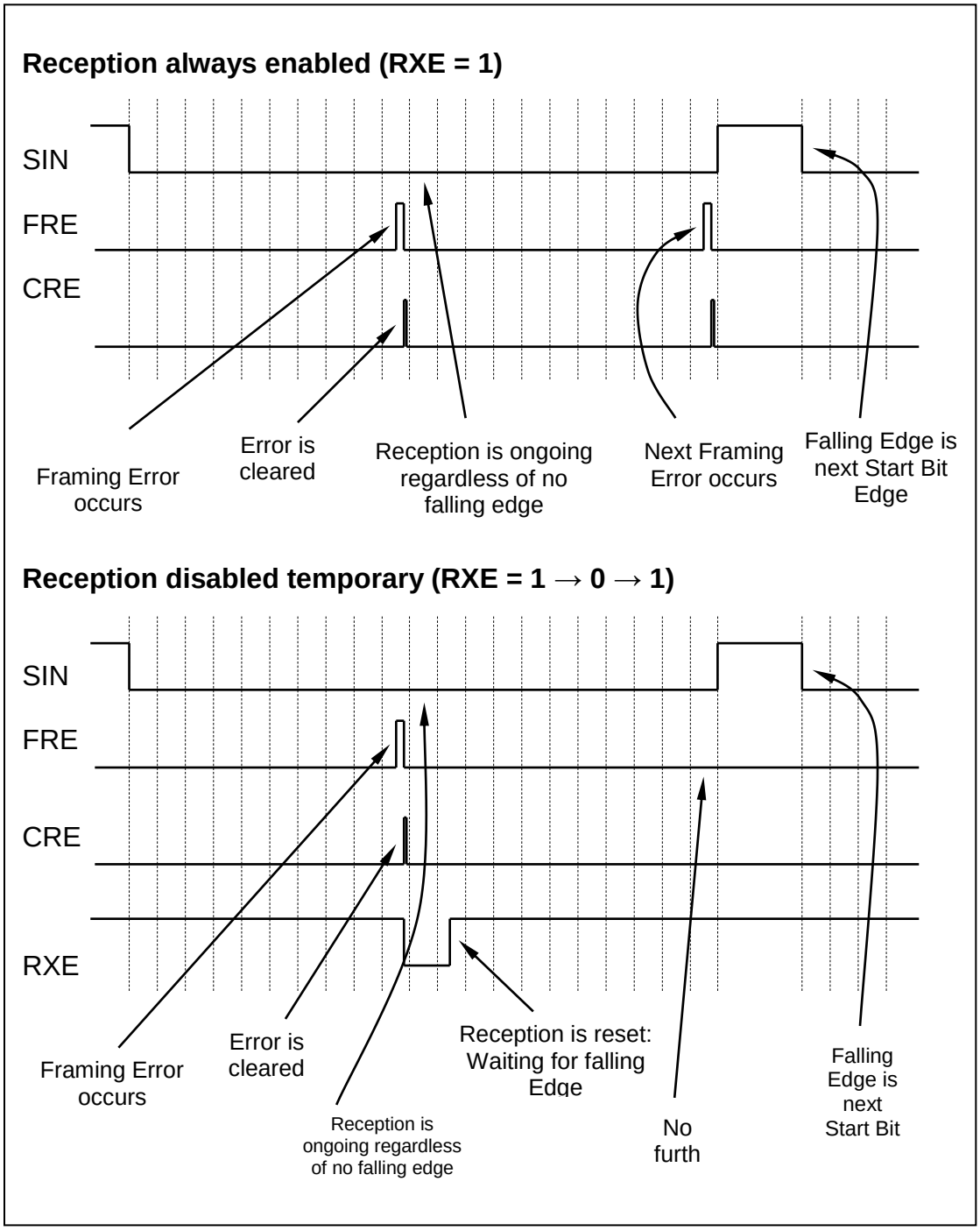
Figure 4. De-synchronization Example



2.4.2 USART Dominant Bus Behavior

Please note that in case a framing error occurred (stop bit: SINn = "0") and next start bit (SINn = "0") follows immediately, this start bit is recognized regardless of no falling edge before as shown in 5. This is used to remain UART synchronized to the data stream and to determine bus always dominant errors by producing next framing errors, if a recessive stop bit is expected. If this behavior is not expected, please disable the reception temporarily (RXE = 1 -> 0 -> 1) after framing error. In this case, reception goes on at next falling edge on SINn.

Figure 5. USART Dominant Bus Behavior



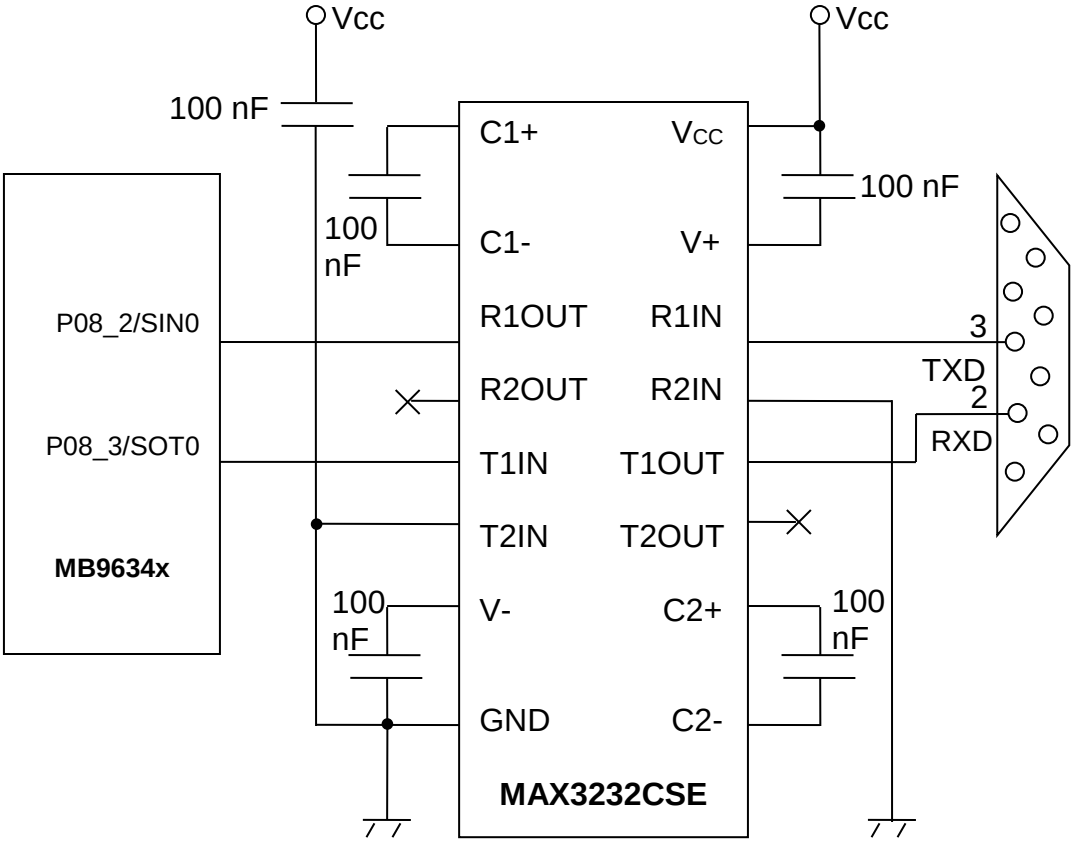
2.5 Interface to the BUS

2.5.1 RS232

USART in asynchronous mode can be interfaced to RS-232 bus via a transceiver. Transceiver provides the ability to receive and transmit the messages over the bus. Figure 6 shows interfacing of MB9634x microcontroller to Transceiver MAX3232CSE. The R1IN input and T1OUT output of the transceiver is connected to TXD and RXD signals of the DB-9 Connector respectively.

The value of the capacitors used is dependent on the supply voltage V_{CC}. Please refer the datasheet of MAX3232CSE for the same.

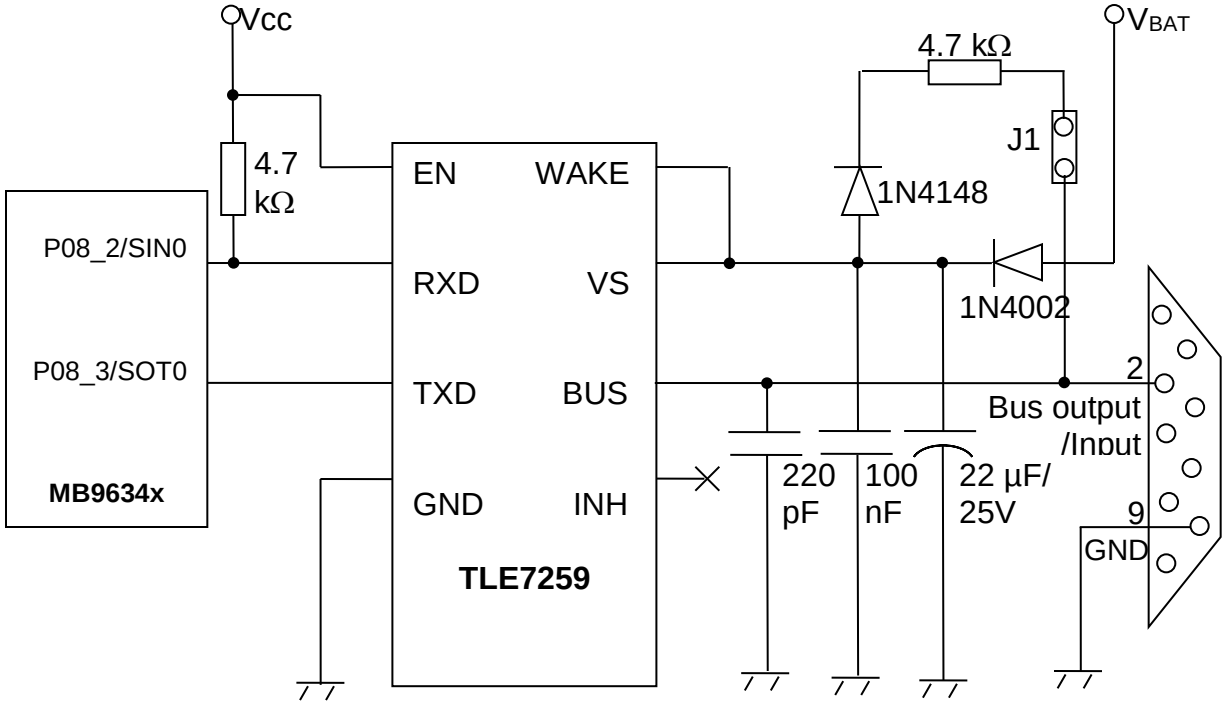
Figure 6. USART Interface to RS-232 Bus



2.5.2 LIN

USART in LIN mode can be interfaced to LIN bus via a transceiver. Transceiver provides the ability to receive and transmit the messages over the bus. Figure 7 shows interfacing of MB9634x microcontroller to Transceiver TLE7259. The BUS Output/Input of the transceiver is connected to Bus Input/output signal of a DB-9 Connector, which is the usual connection of the Cypress Starter kit Boards. V_{BAT} supply needs to chosen within a range of 8V to 18V. Jumper J1 needs to be closed in case of LIN-Master and open in case of LIN-Slave.

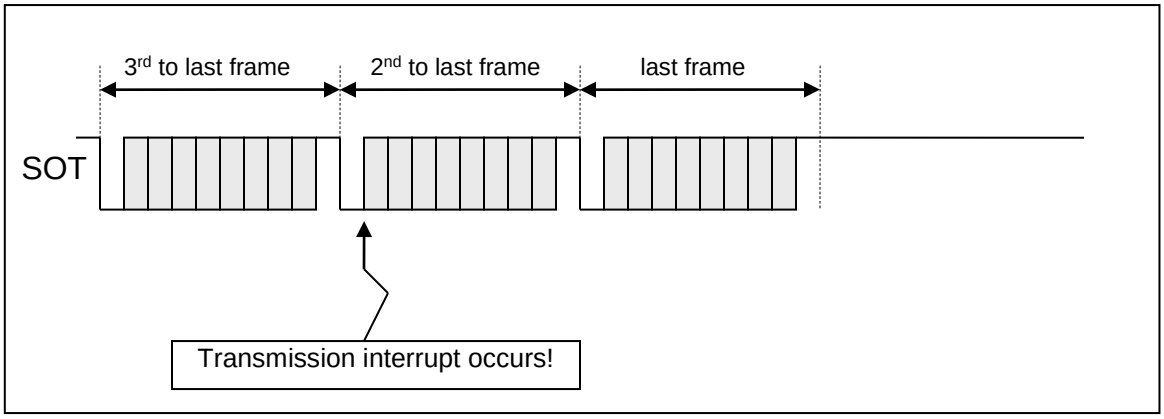
Figure 7. USART Interface to LIN Bus



2.6 DMA Transmission with USART

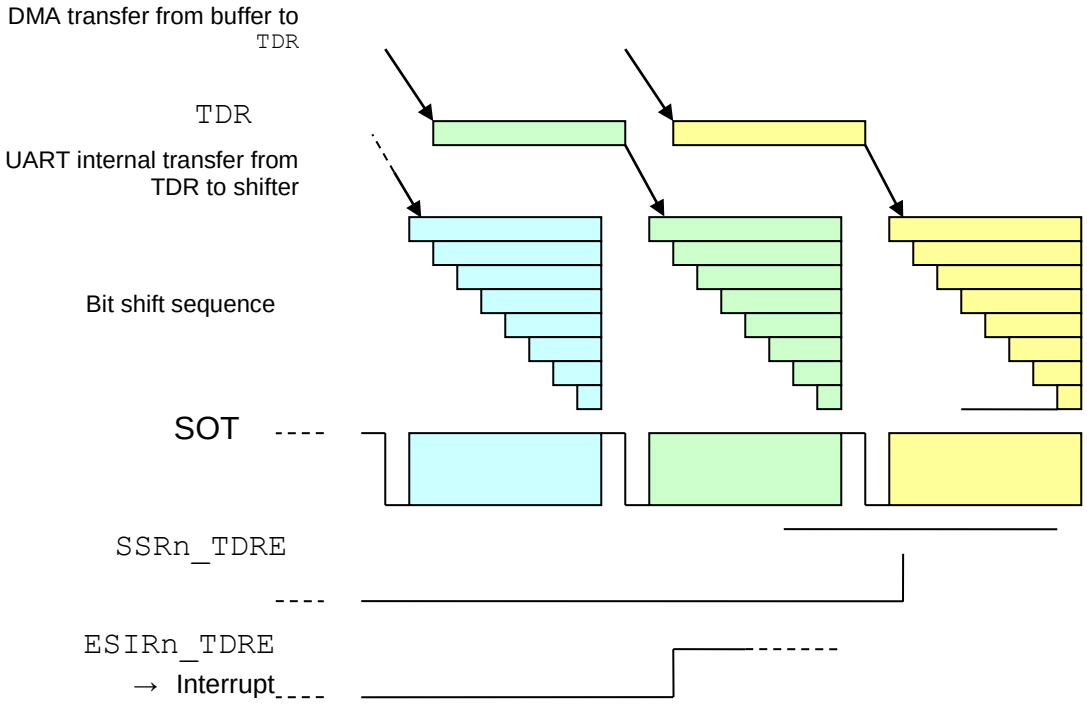
For transmission via DMA the user has to consider the functionality of the transmission data register empty flag (TDRE) timing of the USART. The DMA-end interrupt occurs during the second to last frame of the DMA block. The following graphic illustrates this:

Figure 8. DMA ending transmission interrupt



The time when the interrupt occurs is, when the DMA circuit transfers the last frame to the Transmission Data Register (TDR) of the USART while the 2nd to last frame is being shifted out in the internal shift register of the USART.

Figure 9. DMA ending signals



The ESIRn_TDRE signal, which is selected by ESIRn_AICD = 1 for DMA transfer, generates the transmission interrupt, because at this time the last data (yellow) was transferred to the TDR by DMA.

Caution:
It is not allowed to re-initialize DMA transfer for transmission between ESIRn_TDRE == 1 and SSRn_TDRE == 0 phase, without clearing ESIRn_TDRE!

The phase between ESIRn_TDRE is '1' and SSRn_TDRE is '0' is time critical. Therefore choose a high priority for the USART transmission interrupt and disable globally all interrupts when checking SSRn_TDRE flag and clearing ESIRn_TDRE.

The code example in the appendix (3.4) Shows a consecutively DMA transmission which can be controlled by dma_start variable.

- dma_start = 0; current DMA transfer is finished and then stopped.
- dma_start = 1; DMA transfer is started or continued respectively.

3 USART Examples

Examples for USART

3.1 Asynchronous Mode (0) without Interrupts

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/

void InitUsart0(void)
{
    PIER08_IE2 = 1;              // Enable SIN0 Port Pin
    BGR0 = 1249;                 // 19200 Baud @ 24MHz
    SSR0 = 0x00;                 // LSB first, TDRE bit read only default 1 and written 0
    SMR0 = 0x0d;                 // enable SOT0, reset, Asynchronous normal mode
    SCR0 = 0x17;                 // 8N1, clear possible errors, enable RX, TX
}

void Putch0(char TXchr)         // sends a char
{
    while (SSR0_TDRE == 0);     // wait for transmit buffer empty
    TDR0 = TXchr;               // put TXchr into transmission buffer
}

unsigned char Getch0(void)      // Waits for and returns incoming char
{
    unsigned char RXchr;

    for(;;)
    {
        while(SSR0_RDRF == 0)  // Wait for data received
        {
            __wait_nop();
        }

        RXchr = RDR0;           // Save receive register

        if ((SSR0 & 0xE0) != 0)// Check for errors PE, ORE, FRE
        {
            SCR0_CRE = 1;       // Clear error flags
        }
        else
        {
           return (RXchr);      // Return received character
        }
    }
}
```

Please note, that the function Getch0() only returns a value, if a byte without any reception errors is received. Otherwise this function will never return.

Please also note that the SIN Port Pin has to be enabled. Here it is Port08-2 for MB96 (F) 34x.

3.2 Asynchronous Mode (0) with Interrupts

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.         */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/

#define MAXBUF 100;

unsigned char RXbuf[MAXBUF];      // Reception Buffer
unsigned char TXbuf[MAXBUF];      // Transmission Buffer
unsigned char RXptr = 0;  // Reception Buffer Pointer
unsigned char TXptr = 0;  // Transmission Buffer Pointer

void InitUsart0(void)
{
    PIER08_IE2 = 1;                  // Enable SIN0 Port Pin
    BGR0 = 1249;            // 19200 Baud @ 24MHz
    SSR0 = 0x00;            // LSB first read only default 1 written 0
    SMR0 = 0x0d;            // enable SOT0, reset, Asynchronous normal mode
    SCR0 = 0x17;            // 8N1, clear possible errors, enable RX, TX
}

// Reception Interrupt Service
__interrupt void RX_USART0(void)
{
    if ((SSR0 & 0xE0) != 0)               // Check for errors PE, ORE, FRE
    {
        SCR0_CRE = 1;             // Clear error flags
    }

    if (RXptr < MAXBUF)           // Only, if End of Buffer not reached
    {
        Rxbuf[RXptr] = RDR0;              // Fill Reception Buffer
        RXptr++;                  // Update Buffer Pointer
    }
}

// Transmission Interrupt Service
__interrupt void TX_USART0(void)
{
    TDR0 = TXbuf[TXptr];          // Send Buffer Data
    TXptr++;             // Update Buffer Pointer

    if (TXptr == MAXBUF)          // End of Buffer reached?
    {
        SSR0_TIE = 0;             // Disable Transmission Interrupt
    }
}

// Main Function
void main(void)
{
    InitIrqLevels();
    __set_il(7);                  // allow all levels
    __EI();                       // globally enable interrupts

    // Initialize USART0
    InitUsart0();

    // Fill Transmission Buffer
    . . .

    // Start communication
    SSR0_RIE = 1;         // Enable Reception Interrupts
    SSR0_TIE = 1;         // Enable Transmission Interrupts

    . . .
}
```

Please note, that the transmission starts immediately after the command `SSR0_TIE = 1;` because of the empty Transmission Register (`TDRE = 1`) causing a transmission interrupt.

Please also note that the corresponding interrupt vectors and levels have to be defined in the *vectors.c* module of our standard template project.

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                              */
/*----------------------------------------------------------------------------*/
void InitIrqLevels(void)

    . . .

ICR = (79 << 8) | 6;              // LIN-UART 0 RX of MB96340 Series
ICR = (80 << 8) | 6;              // LIN-UART 0 TX of MB96340 Series

       . . .
}

__interrupt void RX_USART0(void);       // prototype
__interrupt void TX_USART0(void);       // prototype

    . . .

#pragma intvect RX_USART0 79            // LIN-UART 0 RX of MB96340 Series
#pragma intvect TX_USART0 80            // LIN-UART 0 TX of MB96340 Series

    . . .
```

Please also note that the SIN Port Pin has to be enabled. Here it is Port08-2 for MB96(F)34x.

3.3 Synchronous Mode (SPI) Master without Interrupts

The following example shows how to communicate to a NM93CS46 EEPROM. Therefore clock inversion and shift is used (SCES = 1, SCDE = 1). Port PDR09 is used for Chip Select (CS).

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.         */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/

#define DATASIZE 64                  // eeprom memory size in words (16 Bit)

unsigned int data[DATASIZE];         // Data to send to EEPROM
unsigned int readbuffer[DATASIZE];   // Data received from EEPROM

void InitUART(void)
{
        PIER08_IE2 = 1;              // Enable SIN Port Pin
    BGR0 = 15;         // 1M Bit/s @ 16 MHz
    ESCR0 = 0x01;              // SCES = 1 => CPOL = 0
    ECCR0 = 0x10;              // SCDE = 1 => CPHA = 0
    SMR0 = 0x83;               // Mode 2, SCLK enable, SOT enable
    SSR0 = 0x04;               // MSB first, no interrupts
    SCR0 = 0x03;               // Reception and transmission enable
}

void InitPort09(void)
{
        // Bit#2: CS, Bit#1: PE, Bit#0: PRE
        PDR09 = 0x00;                // All Low
        DDR09 = 0x07;                // CS, PE, PRE to output
}

void read_eeprom(unsigned char adr)
{
    unsigned char din, command, dout;

    PDR09_P2 = 1;              // CS = 1

    TDR0 = 0x01;               // Start-Bit (with leading spaces)

    command = (adr & 0x3F) | 0x80;// Address and Write-Instruction
    dout = command;            // Swap Bits for MSB first??
    while (SSR0_RDRF == 0);    // Transmission finished (via reception)?
    din = RDR0;        // Flush reception register
    TDR0 = dout;

    while (SSR0_RDRF == 0);    // Transmission finished (via reception)?
    din = RDR0;        // Flush reception register
    SCR0_CRE = 1;              // Clear possible errors, reset reception state
        //   machine

    // NOTE: Make sure, that SCK is "0" while setting SCDE to "0" (ECCR0 = 0x00;)
    //   In this case (1M bps) no check is needed. Be careful with slower
    //   baud rates!
    ECCR0 = 0x00;              // SCDE = 0 => CPHA = 1 : Needed for special read
        //   timing of used EEPROM (may be not necessary
        //   for other EEPROM)
    TDR0 = 0x00;               // Set dummy byte to produce SCLK

    while (SSR0_RDRF == 0);    // Transmission finished (via reception)?
    din = RDR0;        // MSB
    readbuffer[adr] = (din << 8);

    while (SSR0_TDRE == 0);
    TDR0 = 0x00;               // Set dummy byte to produce SCLK

    while (SSR0_RDRF == 0);
```

```
    din = RDR0;           // LSB
    readbuffer[adr] = (readbuffer[adr] | din);

    ECCR0 = 0x10;                 // SCDE = 1 => CPHA = 0 : Set back for write timing

    PDR09_P2 = 0;                 // CS = 0
}

void write_eeprom(unsigned char adr)
{
    unsigned char dout, command;

    PDR09_P2 = 1;                           // CS = 1

    while (SSR0_TDRE == 0);
    TDR0 = 0x01;                  // Start-Bit (with leading spaces)

    command = (adr & 0x3F) | 0x40;          // Address and Write-Instruction
    dout = command;
    while (SSR0_TDRE == 0);
    TDR0 = dout;

    dout = (data[adr] >> 8) & 0xFF;         // MSB
    while (SSR0_TDRE == 0);
    TDR0 = dout;

    dout = data[adr] & 0xFF;                // LSB
    while (SSR0_TDRE == 0);
    TDR0 = dout;

    while (ECCR0 & 0x01);                   // Wait for start of transmission (or ongoing)
    while (!(ECCR0 & 0x01));                // Wait for transmission finished

    PDR09_P2 = 0;                           // CS = 0

    wait(1);

    PDR09_P2 = 1;                           // CS = 1

        // Next function (waiting for busy release) is made by
    //   polling. Please note, that for the NM93CS46 EEPROM the
    //   wait time can take till 10 ms! I. e. the CPU is then
    //   also busy. For fast application a timer should be used,
    //   which generates an interrupt after 10 ms from here,
    //   so that the CPU can perform other jobs in this time.
        while(ESCR0_SIOP == 1);             // Wait for EEPROM busy
    while(ESCR0_SIOP == 0);                 // Wait for EEPROM busy release
    PDR09_P2 = 0;                           // CS = 0
}

void write_enable(void)
{
    PDR09_P2 = 1;                           // CS = 1

    while (SSR0_TDRE == 0);
    TDR0 = 0x01;                  // Start-Bit (with leading "zeros")

    while (SSR0_TDRE == 0);
    TDR0 = 0x30;                  // WEN command

    while (ECCR0 & 0x01);                   // Wait for start of transmission (or ongoing)
    while (!(ECCR0 & 0x01));                // Wait for transmission finished

    PDR09_P2 = 0;                           // CS = 0
}

void write_disable(void)
{
    PDR09_P2 = 1;                           // CS = 1

    while (SSR0_TDRE == 0);
    TDR0 = 0x01;                  // Start-Bit (with leading "zeros")
```

```
    while (SSR0_TDRE == 0);
    TDR0 = 0x00;                    // WDS command

    while (ECCR0 & 0x01);                   // Wait for start of transm. (or
ongoing)
    while (!(ECCR0 & 0x01));                // Wait for transmission finished

    PDR09_P2 = 0;                   // CS = 0

}
```

Please note, that the SIN Port Pin has to be enabled. Here it is Port08-2 for MB96(F)34x.

3.4 DMA block transmission

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/

void InitUart0(void)
{
    // switch port pins to input / output accordingly (depends on MCU)
    DDR ...
    PIER08 ...

    BGR0 = 486;       //  115200 Baud @ 56 MHz

    SCR0 = 0x17;      // 8N1
    SMR0 = 0x0D;      // enable SOT0, Reset, normal mode
    SSR0 = 0x00;      // LSB first, no interrupt

    ESIR0 = 0x01;     // AICD = 1 for DMA
    SMR0_UPCL = 1;    // set AICD via UART reset for DMA
}

void InitDMA(void)
{
    DSR = 0x0000;     // Clear possible DMA 0 request
    DER = 0x0000;     // DMA disable

    DISEL0  = 80;     // UART0 irq number
    DISEL1  = 0;      // Initialize unused DMA channels to 0
    DISEL2  = 0;
    DISEL3  = 0;

    DCTH0 = 0;
    DCTL0 = (MAXBUFFER - 1);              // # of Bytes
    IOAH0 = (unsigned char) &TDR0 >> 8;         // I/O Bank 00
    IOAL0 = (unsigned char) &TDR0 & 0xFF;
    DMACS0 = 0x12;    // no IOA update, BAP update, byte transfer, BAP -> IOA
    BAPH0 = (__far unsigned long) &buffer[0] >> 16;
    BAPM0 = (__far unsigned long) &buffer[0] >> 8;
    BAPL0 = ((__far unsigned long) &buffer[0] & 0xFF);

    DER_EN0 = 1;      // DMA 0 enable
}

__interrupt void IrqUART0tx (void)
{
    SSR0_TIE = 0;     // disable tx interrupt
    DSR_DTE0 = 0;     // Clear DMA request

    __wait_nop();     // delay for leaving ISR not too early
       __wait_nop();

    dma_start = 0;    // announce DMA end
}
```

Please note that `MAXBUFFER` and `unsigned char buffer [MAXBUFFER]` has to be defined by the user itself.

Please also note that the USART interrupt should have a high priority.

```
void main(void)
{
unsigned char dma_start = 0;

. . .

   if ((“some need for DMA transfer”) && (dma_start == 0))
   {
       __DI();       // disable Interrupts globally

       // Phase within ESIR0_TDRE == 1 and SSR0_TDRE == 0?
       if (SSR0_TDRE == 0)
       {
           ESIR0_TDRE = 0;
       }

       dma_start = 1;
       InitDMA();
       SSR0_TIE = 1;        // enable UART tx "interrupt" for DMA

__EI();       // enable Interrupts globally
   }

   . . .

}
```

4 APPENDIX A

Related Documents

4.1 Related Documents

Please find further information in the following documents.

- AN205253 - F2MC-8L/16LX/FR Family with LIN-USART
- AN205498 - F2MC 16FX Family, USART

Note: That the above mentioned documents are written for 16LX MCUs, but also usable for 16FX-USART. Please also note that for 16FX devices the serial data or clock input port pin has to be enabled via PIER register.

5 Additional Information

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

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

The software example related to this application note is:

96340_uart0_async

96340_uart0_async_interrupt

96340_uart1_async

96340_uart2_async

96340_uart3_async

96340_uart_lin_master

96340_uart_lin_slave

96340_uart_sync_spi_nm93cs46

96340_dma_uart0

It can be found on the following Internet page:

http://www.cypress.com/products/16FX

6 Document History

Document Title: AN205498 - F²MC 16FX Family, USART

Document Number: 002-05498

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	04/28/2006	Initial Release
			12/07/2006	Reviewed the document and updated with review findings
			02/21/2007	Updated with re-review findings Added clock modulator related information
			08//29/2007	Added schematic, error conditions and updated register related information
			10/10/2007	Timing diagrams updated
			07/07/2008	Added information on PIER
			02/19/2010	DMA transmission sub chapter and example code added
			06/11/2010	CPOL, CPHA logic corrected
*A	5068980	MKEA	12/31/2015	Migrated Spansion Application Note MCU-AN-300205-E-V17 to Cypress format
*B	5838989	AESATMP8	07/31/2017	Updated logo and Copyright.
*C	6038734	NOFL	01/20/2018	Updated logo and links. Updated Sales page and Copyright year.

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 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2006-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.