Freescale Semiconductor

Order this document by
AN1667/D

AN1667/D

SOFTWARE SCI IMPLEMENTATION TO THE MISC COMMUNICATION PROTOCOL

By Jaromir Chocholac
Roznov Systems Application Laboratory
FREESCALE - Czech Republic

1 INTRODUCTION

This application note describes a software implementation of asynchronous serial communication for microcontroller units which do not possess a hardware SCI channel, such as very low-cost microcontrollers (MCUs). Using these MCUs is interesting in low cost Niche Area Networks. To perform asynchronous serial communication in such MCUs, software emulation of the serial communication must be used. Previous software SCI solutions have only provided very basic communications. What is remarkable about the system described here is that a complete data link protocol stack is implemented and executed. The implementation of the MISC communication protocol is presented here as an example.

2 OVERVIEW

The hardware SCI module provides full-duplex asynchronous serial communication between two devices. Full duplex means that the device is able to transmit and receive at the same time. The SCI handles all communications duties and then the CPU can do other functions simultaneously. The hardware SCI receiver can detect error conditions automatically, such as framing, noise, and overrun. In SCI software emulation it is much easier to perform only half-duplex communication. Half-duplex means, that device cannot receive while it is transmitting, and it cannot transmit while it is receiving. Then, in the software SCI emulation, the error detection has to be done by the receiver part of software. The solution described here is dedicated for the implementation MUX wiring protocols. Most of these protocols are using full duplex communication, to have easy bus error detection ability. Virtual full-duplex communication is implemented for this MISC communication protocol emulation.

The requirements of this software emulation are:

- Keep features close to hardware SCI module
- Ability to detect IDLE state on the line
- Ability to detect noise, framing and overrun errors while receiving
- Using only basic MCU’s modules (I/O pins, IRQ pin, Core timer)
- Easy implementation of MUX wiring protocols

3 TRANSMISSION FORMAT

Standard asynchronous communication uses non-return-to-zero (NRZ) format consisting of one start bit followed by eight or nine data bits and one or two stop bits. If there is no communication, the line is high (logic one). If the line remains high for more than ten or eleven bit periods, it is said to be in the “IDLE” state. Data is both transmitted and received, least significant bit (LSB) first. Each bit has a duration (t_p.) which defines the baud rate. We can define t_p as a bit period by the equation:

$$t_p=1/\text{Baud Rate} \qquad \text{(EQ 3-1.)}$$

Transmission or reception always begin by the transition of the line from high to low and staying there for one bit period t_p (start bit). Because the data being received are asynchronous to the device's programs this start negative edge may be use to synchronize to the communication program through the IRQ pin. Figure 3-1 shows the NRZ Transmission format.

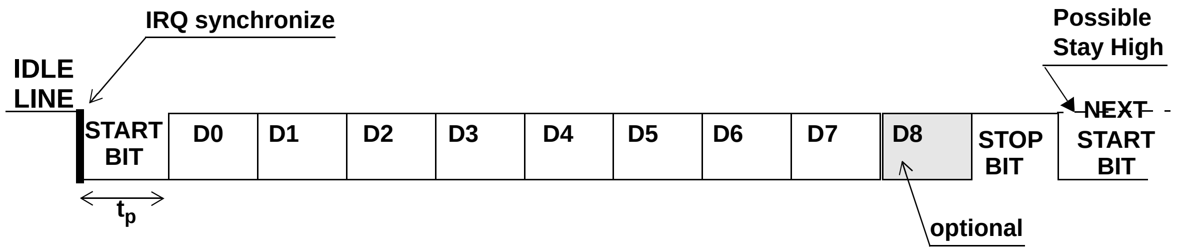

Figure 3-1. NRZ Transmission Format

3.1 Data Sampling Technique

The SCI receiver software routine samples each bit three times in the middle of each bit period. Figure 3-2 shows the SCI receiver sample points. The sampling interval (t_s) between sampling points is independent of the Baud Rate, it depends on the CPU bus frequency (f_{bus}). We define t_s by the equation:

$$t_s = 7/f_{bus} \qquad \text{(EQ 3-2.)}$$

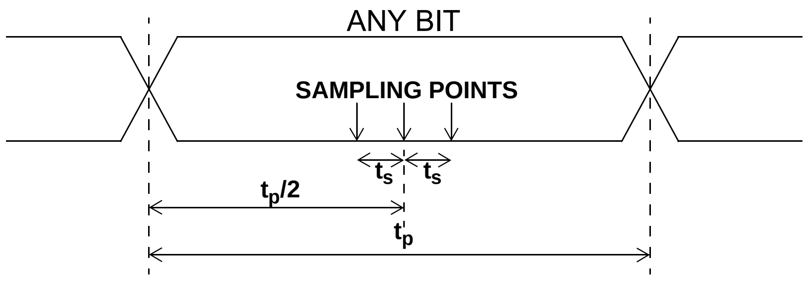

Figure 3-2. SCI Receiver Sample Points

3.1.1 Start bit detection

The SCI receive function is not easy due to the asynchronous nature of incoming serial data. The key point of this is to recognize a start bit because this procedure determines the amount of Baud Rate frequency mismatch that can be tolerated. The start bit detection used in the discussed SCI software implementation is not so complex as in the hardware SCI, but it meets the requirements for 9600 Baud. The negative edge of the start bit is used to synchronize the communication program through the IRQ pin. The IRQ interrupt service routine than samples data on the RxD pin three times in the middle of the setup bit period. If two samples are different from zero, the start edge is rejected and the interrupt service routine is ended.

3.1.2 Noise detection

Perceived noise in any of the data bit periods or the start or stop bit periods will cause the Noise Flag (NF) to be set. During each bit period, including the start and stop, data samples are taken near the middle of the bit period. (see Figure 3-2). If any sample disagrees with the rest, the working NF is set.

3.1.3 Frame error detection

The stop bit is defined as a logic one. If a logic zero is detected where the stop bit was expected, the Frame Error (FE) flag is set.

3.1.4 Overrun error detection

Since the SCI receiver software routine is using a double buffer, there is a full character time between reception of a character and when it must be read from the SCI Data Register (SCDR) to avoid an OverRun (OR) flag setting caused by a subsequent character. In an OR condition, the character that caused the OR is lost, but the previously received character in the SCDR is not disturbed.

4 MCU AND MUX BUS HARDWARE DESCRIPTION

For SCI emulation I/O pins are used as the transmit data (TxD) and receive data (RxD) pins. The RxD pin is connected to the IRQ pin to ensure communication synchronization. The basic MUX wiring node, from the communication point of view, consists of an MCU and a MUX bus Physical layer. The Physical layer converts TxD and RxD signals to the single wire MUX bus. In the MISC communication protocol, the transmitting MCU is receiving the same data as it is transmitting. Data from the MCU's TxD pin are mirrored to the MCU's RxD pin through the Physical layer. This method is good for a virtual full-duplex software emulation. Figure 4-1 shows basic MUX wiring schematic.

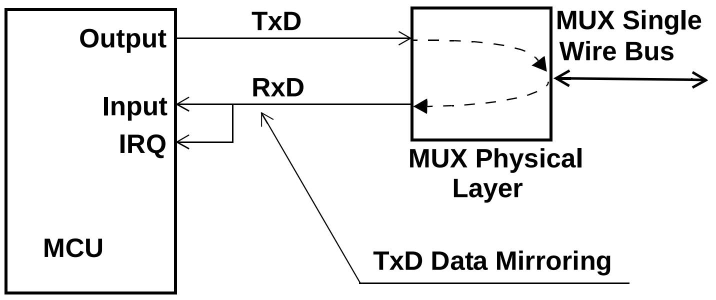

Figure 4-1. Basic MUX Wiring Schematic

What does virtual full-duplex software emulation mean? When the MCU transmits data out from the TxD pin to the Physical layer it should be able to read the mirrored data from the RxD pin. The serial data bit transfer is fully under software control. When data is transmitting out, there is some time inside each bit period, when data on the TxD pin is stable and the MCU is waiting for the end of the bit period. This time can be used to read data back from the RxD pin, check it for noise and store it as a received frame. Noise detection is based on the same principle as shown in Figure 3-2.

5 SOFTWARE MODEL

It is easy to modify the hardware based SCI LMAC to the software SCI version if the register features remain close. There are, in this software SCI emulation, four memory locations (SW_SCCR1, SW_SCCR2, SW_SCSR, SW_SCDR). to control and monitor the software SCI operation. These registers have the same functions and flags as hardware SCI registers.

There is one small exception in the SW_SCSR register in bit LSB, where in the software SCI implementation it has the meaning Last Access (LA). When the RxD pin has been busy and has received data, the LA flag is set. Only after the LA flag has been set, can the IDLE character detection be started. The IDLE character detection is necessary to be built-in, due to the requirements of MUX wiring protocols. This function is supported by the core timer interrupt software routine.

Beside these basic registers there are two more memory locations, which emulate the shift registers. One of them is used by the transmitter part and the second one by the receiver part of the software.

TX_SHIFT_REG - Software SCI transmitter shift register

The Software SCI transmitter shift register is used by the transmitter software routine for parallel to serial data conversion. The SCI transmitter software routine is double buffered, which means that the transmitter software routine can have up to two characters in it at any given moment. While one of the characters is in the data buffer (SW_SCDR), another could be shifted out from the transmitter shift register (TX_SHIFT_REG) to the TxD pin.

RX_SHIFT_REG - Software SCI receiver shift register

The Software SCI receiver shift register is used by the receiver software routine for serial to parallel data conversion. The SCI receiver software routine is double buffered, which means that the receiver software routine can have up to two characters in it at any given moment. One of the characters is in the data buffer (SW_SCDR), another could be shifted into the receiver shift register (RX_SHIFT_REG) from the RxD pin. This double-buffered arrangement gives software some time to notice a received character and read it before the next serial character is finished.

6 SOFTWARE DESCRIPTION

The Assembler source code file SW_SCI_M.ASM is dedicated for MUX wiring applications with virtual full duplex emulation (see Figure 6-1). The SCI software consists of two main subroutines, the **PUT_byte** for transmission and the **GET_byte** for receiving. All interrupts must be disabled when sending or receiving a byte. The transmitting routine, **PUT_byte**, can be called by the main program or by the core timer interrupt service routine. This **PUT_byte** routine transmits serially the contents of the TX_SHIFT_REG using the transmit data line TxD. Figure 6-1 shows the **PUT_byte** routine state diagram.

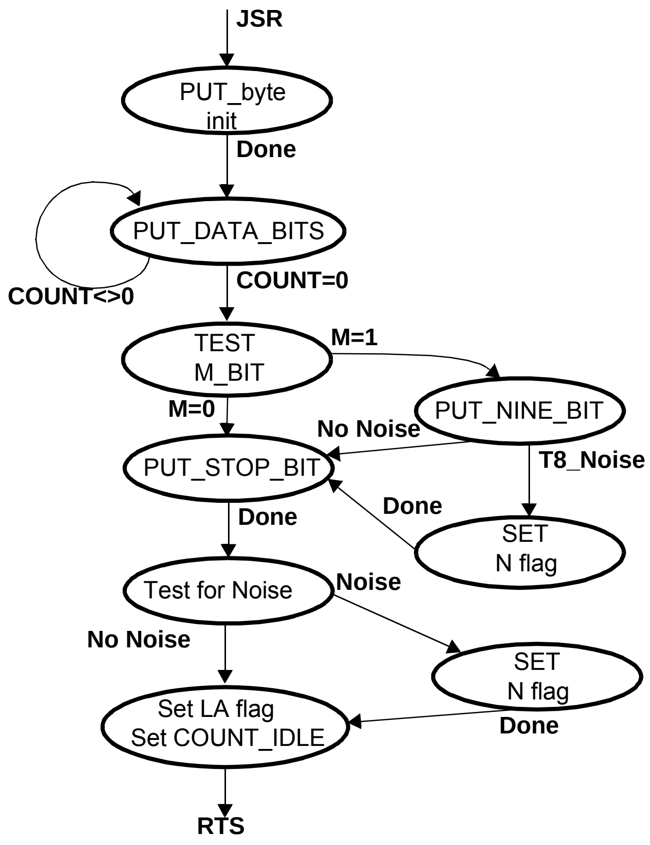

Figure 6-1. PUT_byte State Diagram

Inside the PUT_DATA_BITS, PUT_STOP_BIT and PUT_NINE_BIT states, the RxD pin is sampled by the **GET_BIT** subroutine**,** to check that the bus line does not have an error condition. The **PUT_byte** software routine controls the TDRE, TC, RDRF, NF, FE, OR, LA and IDLE flags. The **PUT_byte** software is controlled by the TE and M flags.

The **GET_byte** receive routine, receives one byte of data from the receive data line RxD and places it into the SW_SCDR. The **GET_byte** receive routine has to be called by the IRQ interrupt service routine to receive a start bit and the subsequent frame. Figure 6-2 shows the **GET_byte** routine state diagram. Because all timing is done by software, the CPU will remain in **GET_byte** until after the full character is received.

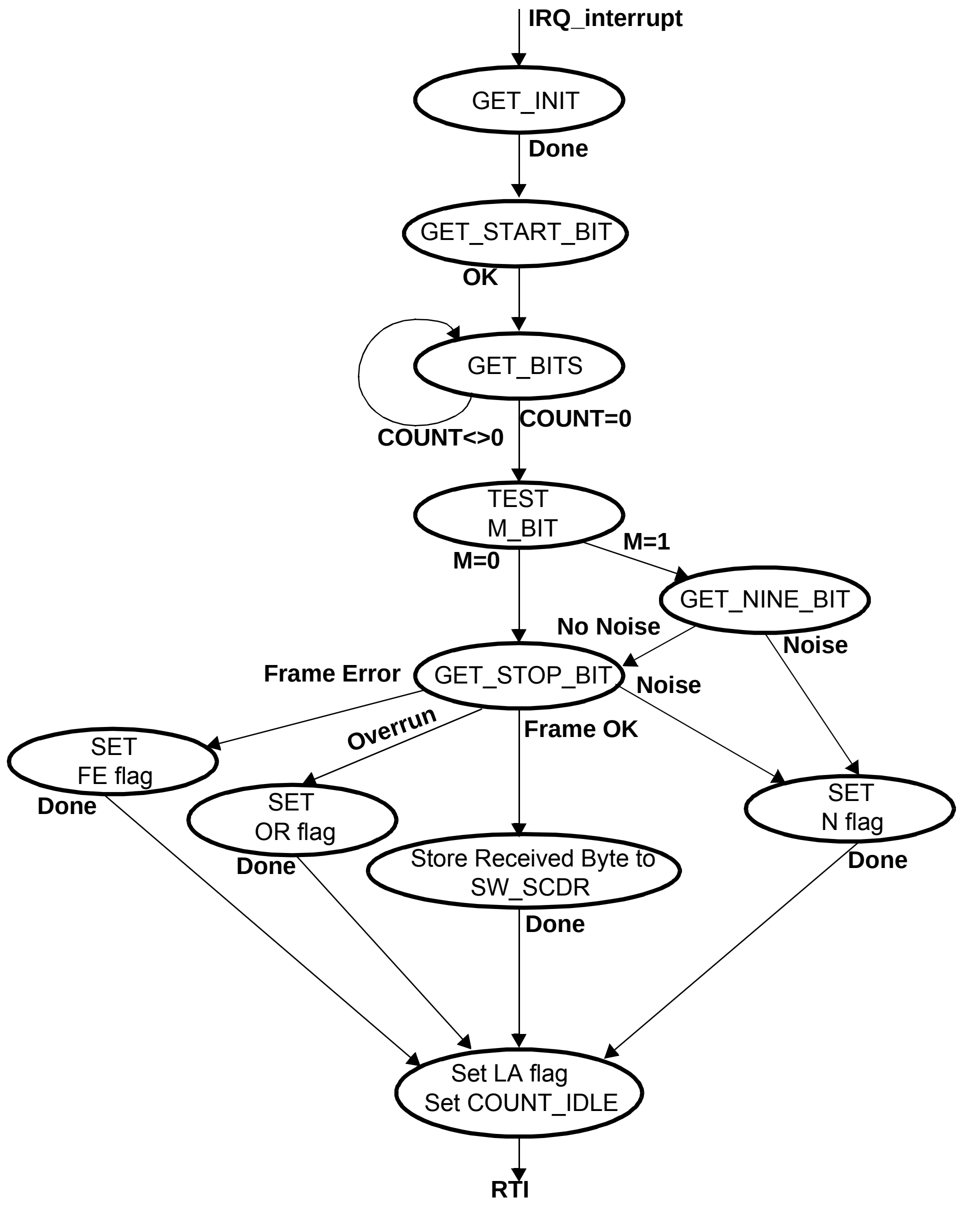

Figure 6-2. GET_byte State Diagram

Inside the GET_START_BIT, GET_BITS, GET_STOP_BIT and GET_NINE_BIT states, the RxD pin is sampled by the **GET_BIT** subroutine**,** to check that the bus line does not have an error condition.The **GET_byte** software routine controls the RDRF, NF, FE, OR, LA and IDLE flags. The GET_byte software routine is controlled by the RE and M flags.

The search for IDLE, **S_IDLE,** software routine checks the RxD line for the IDLE character and controls the IDLE flag. The **S_IDLE** routine has to be called by the core timer interrupt service routine. This interrupt routine is servicing interrupts from the Timer Overflow flag (TOF). Once IDLE has been cleared, it cannot be set again until the RxD line has been busy and then becomes idle again.

The **READ_SCDR** routine reads the SW_SCDR register and clears flags in the SW_SCSR.

The **SW_SCI_INIT** routine has to be called at the beginning of the application program. It sets the initial content of the software SCI registers.

6.1 Search for IDLE

The idle condition is defined as at least a full character time of logic one on the RxD line. A frame time is 10 bit periods if M = 0 or 11 bit periods if M = 1. The search for IDLE, **S_IDLE,** software routine checks the RxD line for the IDLE character and controls the IDLE flag. The **S_IDLE** routine has to be called by the core timer interrupt service routine. This interrupt routine services interrupts from the Timer Overflow flag (TOF). The example described here is done for the microcontroller HC05J1A with internal bus frequency 2 MHz. The SCI software is optimized for a 2MHz bus frequency and 9600 Baud rate for communication but it can be easily changed for other requirements. For this purpose, there is the BAUD_SEL constant in the SW_SCI.h file, which can be changed according to the new conditions.

For a 2 MHz bus frequency the TOF flag is set every 512us. Figure 6-3 shows the relationship between the RxD line, Core Timer and S_IDLE routine. The Y axis represents the value of the first eight stages of the core timer. It can be read from the Timer Counter Register (TCR).

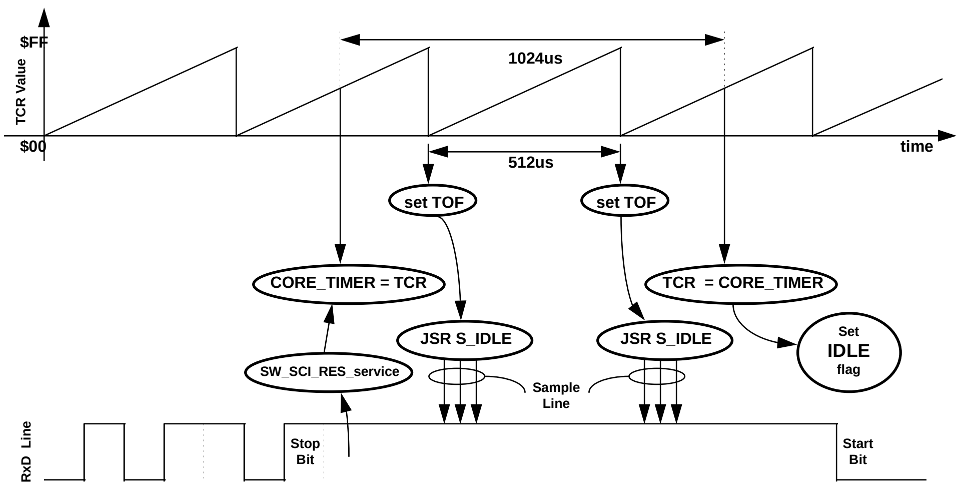

Figure 6-3. Relationship Between Data Line, Core Timer and S_IDLE Routines

The **S_IDLE** routine uses the **COUNT_IDLE** variable, for counting the number of the timer overflows and the **CORE_TIMER** variable for the TCR initial value store. The **COUNT_IDLE** variable is initialized by the **SW_SCI_INIT** routine according to the Baud rate selection to the value BAUD_SEL/4. The **SW_SCI_RES_service** routine initializes the **CORE_TIMER** variable. The **S_IDLE** routine checks the RxD line for logic one. If the RxD line is in the logic one state then the **S_IDLE** routine decrements the **COUNT_IDLE** variable and, if its value is zero, the IDLE flag is set. In the case that the RxD line is zero, the IDLE flag is cleared. Once IDLE has been cleared, it cannot be set again until the RxD line has been busy and then becomes idle again. The LA flag in SW_SCSR register indicates the last access on the RxD line, it means that RxD pin has been busy and data was received. In this case the LA flag is set. The LA flag is checked at the beginning of the **S_IDLE** routine. This **S_IDLE** routine seems to be not so precise, due to rare sampling of the data line, but it is sufficient because of the master slave communication technique.

7 INTERRUPT SERVICES

The user has to design some interrupt routines to help the communication hardware and software. The Real time interrupt service routine is servicing interrupts from the Timer Overflow Flag (TOF) and interrupts from the Real Time Interrupt Flag (RTIF). If the TOF flag is set and the communication software is in the receiving state (LMAC_SEND=0), then the **S_IDLE** subroutine must be called. In the case that the RTIF flag is set, the Data Link service must be done and any user timer service can be done too.

The IRQ interrupt service routine services the frame receive function. The **GET_byte** subroutine must be called. When **GET_byte** subroutine is done the SCI flags must be tested and then, according to the flags, the UMAC_service has to be performed or not. The state diagram of the interactions between the interrupt routines is shown in Figure 7-1.

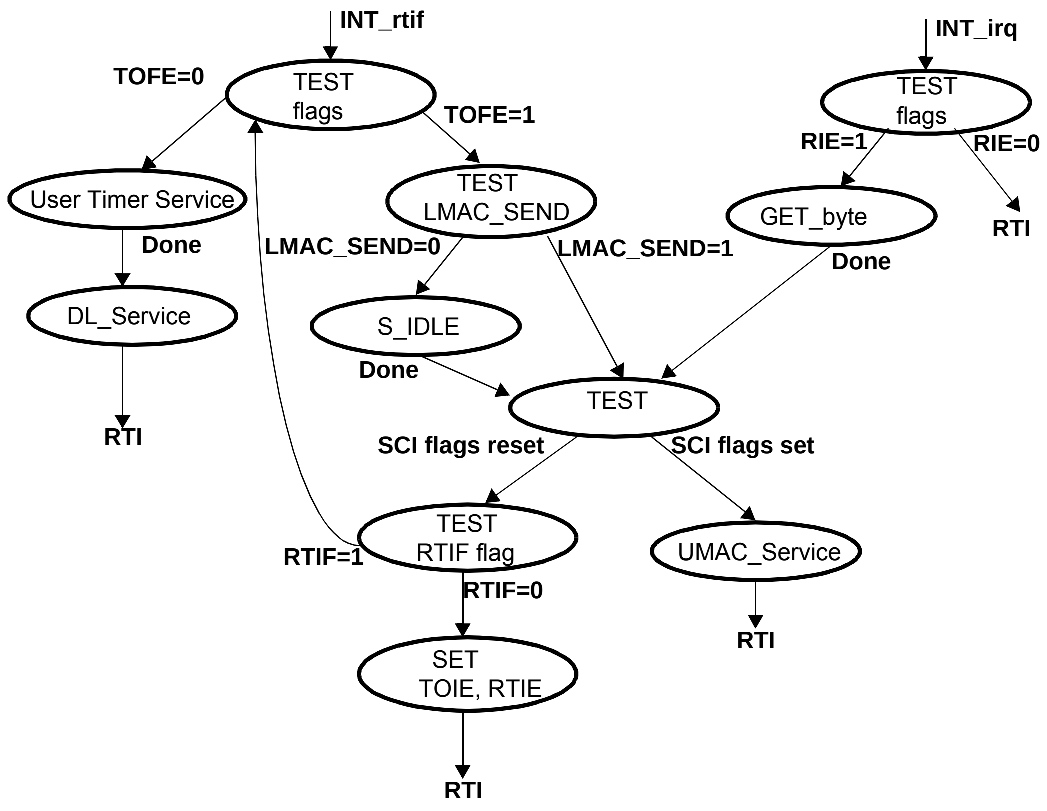

Figure 7-1. Interrupt Service Routines State Diagram

8 IMPLEMENTATION OF THE MISC PROTOCOL

The basic aim of the software SCI implementation of the MISC communication protocol was, to keep the software MISC communication library files as close as possible to the hardware MISC communication library files. The low level layer of communication protocol (LMAC) is device dependent so the main changes have to be done here. The LMAC_RESET_request routine has to initialize the software SCI channel. The SW_SCI_RES_service routine was added here for the software SCI initialization.

Each message in the MISC protocol is standardized and comprises two fields. The Master sends a Push field and then the selected slave responds by a Pull field. See Figure 8-1. MISC Message Composition. The Slave is receiving the Push field and sending the Pull field. In the software SCI implementation it is necessary to have different condition settings for the push field receiving and pull field sending. If the Slave is receiving a Push field, the IRQ interrupt must be enabled to detect the start bit. The Core Timer Overflow interrupt must be disabled to prevent the loss of the next byte, in case the interrupt from the TOF flag occurs during IFS.

If the Slave is sending a Pull field, the IRQ interrupt must be disabled because the reading back of the RxD pin is controlled by the PUT_byte routine. The Core Timer Overflow interrupt must be enabled to start IDLE character detection during IFS. Then the LMAC_SEND flag was added to the LMAC_status variable to indicate if the Slave is sending or receiving.

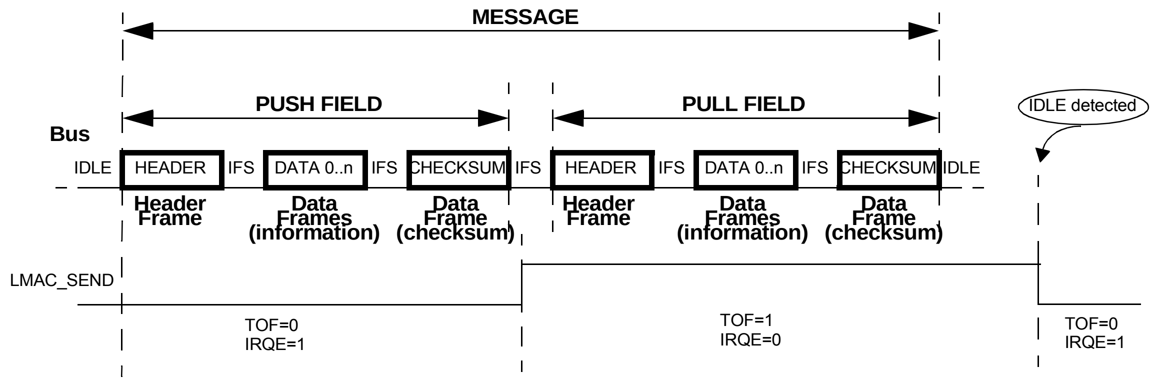

Figure 8-1. MISC Message Composition

This LMAC_SEND flag is driven by the SEND_FIELD_ENA and REC_FIELD_ENA routines as well as other necessary conditions. The software SCI control routines are explained in the next sections.

8.1 Software SCI Control Routines Description

The device oriented control routines are included in the SW_SCI_S.INC file. These routines modify the DLLMAC and DLUMAC service routines.

SEND_FIELD_ENA (sending field enable routine).

This routine services the condition settings for the pull field sending. If the Slave is sending a Pull field, the IRQ interrupt must be disabled because the reading back of the RxD pin is controlled by the PUT_byte routine. The Core Timer Overflow interrupt must be enabled to start IDLE character detection during IFS. The LMAC_SEND flag in the LMAC_STATUS register is set to a logic one to indicate this state.

This routine performs:

- IRQ interrupt disabling
- LMAC_SEND flag in LMAC_STATUS setting
- Core Timer Overflow interrupt enabling

REC_FIELD_ENA (receiving field enable routine).

This routine services the condition settings for the push field receiving. If the Slave is receiving a Push field, the IRQ interrupt must be enabled to detect the start bit. The Core Timer Overflow interrupt must be disabled to prevent the loss of the next byte, in case the interrupt from the TOF flag occurs during IFS. The LMAC_SEND flag in the LMAC_STATUS register is cleared to a logic zero to indicate this state.

This routine performs:

- IRQ interrupt enabling
- LMAC_SEND flag in LMAC_STATUS clearing
- Core Timer Overflow interrupt disabling

LMAC_TOF_OFF (core timer overflow interrupt disable routine).

This routine services the condition settings for the receiving after the IDLE was received. If the IDLE was received, just the IRQ interrupt must stay enabled to detect the start bit and the Core Timer Overflow interrupt must be disabled.

This routine performs:

- Core Timer Overflow interrupt disabling

SW_SCI_RES_service (SW_SCI reset initialization routine).

This routine services the initial condition settings for the software SCI.

This routine performs:

- IRQ interrupt request bit clearing
- IRQ interrupt enabling
- Core timer actual value storing
- LMAC_SEND flag in LMAC_status clearing

9 CONCLUSION

This application note describes a software implementation of asynchronous serial communication for microcontroller units without a hardware SCI channel. Using these MCUs is interesting in low cost Niche Area Networks. To perform asynchronous serial communication, software emulation of the SCI is used. What is remarkable about the system described in this application note is that a complete data link protocol stack is implemented and executed. The use of a low cost micro-controller will enable many other applications in the Industrial and White Goods field to be realized.

The software source files (SWITCH.ZIP) you can find on the http://design-net.sps.mot.com.

REFERENCES

AN1240/D - HC05 MCU Software-Driven Asynchronous Serial Communication Technique Using MC68HC705J1A

NOTES

NOTES

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN1667/D