NeoRS232_AVR Reference Manual

Generated by Doxygen 1.4.2

Sun Nov 27 18:41:00 2005

Contents

1	AVI	R_NeoRS232	1
2	Neo	RS232_AVR Class Index	3
	2.1	NeoRS232_AVR Class List	3
3	Neo	RS232_AVR File Index	5
	3.1	NeoRS232_AVR File List	5
4	Neo	RS232_AVR Class Documentation	7
	4.1	COM_HANDLE_T Struct Reference	7
	4.2	FRAMER Struct Reference	9
5	Neo	RS232_AVR File Documentation	11
	5.1	CRC_Test.c File Reference	11
	5.2	define.h File Reference	13
	5.3	main.c File Reference	18
	5.4	serieData.c File Reference	19
	5.5	simpleRead.c File Reference	20
	5.6	simpleWrite.c File Reference	21
	5.7	timer.c File Reference	22
	5.8	usart c File Reference	23

AVR_NeoRS232

This project implements an Idle RQ Stop & Wait protocol. This is a complicated manner to say that a transmitter must wait for an answer of the receiver before it can send something again. The receiver checks the incoming frame using the CRC16-checksum. If the checksum is correct, then an acknowledge is sent back. In the meanwhile, the sender had started a timer after sending his frame. It waits for the acknowledge of the receiver. If that acknowledge doesn't arrive in five seconds, then it tries to resend the data.

A frame consists of a startflag (0xAA), the data and a stop flag (0xFF). If a 0xFF (or some other flag character) occurs in the data, the receiver would falsely think that the end of the frame has been reached. To prevent this, bytestuffing is used. Before every flag character in the data we send the DLE-flag.

When the receiver sees the DLE-flag, it knows that the next character must be considered as a databyte and not as a flag byte.

Oh, remember: to use this package, you must have some version of javacomm installed. It's available from the Java website.

Author:

LieBtrau

locomotieff@tiscali.be

Version:

version 1.1

2 AVR_NeoRS232

NeoRS232_AVR Class Index

2.1 NeoRS232 AVR Class List

Here are the classes, structs, unions and interfaces with brief descriptions:	
COM_HANDLE_T	
FRAMER	

NeoRS232_AVR File Index

3.1 NeoRS232_AVR File List

Here is a list of all documented files with brief descriptions:

CRC_Test.c .							 															1
define.h							 															1
main.c							 															1
serieData.c .							 															1
simple Read.c																						
simpleWrite.c																						
timer.c							 															2
usart.c							 															2

NeoRS232_AVR Class Documentation

4.1 COM_HANDLE_T Struct Reference

Public Attributes

- uint8_t * pRxPut
- uint8_t * pRxGet
- uint8_t * pRxBegin
- uint8_t * pRxEnd
- uint8_t rxCount

4.1.1 Detailed Description

Struct for working with the circular buffer for receiving characters.

4.1.2 Member Data Documentation

4.1.2.1 uint8_t* COM_HANDLE_T::pRxBegin

begin of RX-buffer

4.1.2.2 uint8_t* COM_HANDLE_T::pRxEnd

end of RX-buffer

4.1.2.3 uint8_t* COM_HANDLE_T::pRxGet

pointer to get next received byte

4.1.2.4 uint8_t* COM_HANDLE_T::pRxPut

pointer to put next received byte

4.1.2.5 uint8_t COM_HANDLE_T::rxCount

number of unread bytes received

The documentation for this struct was generated from the following file:

• usart.c

4.2 FRAMER Struct Reference

```
#include <define.h>
```

Public Attributes

- uint8_t * pBegin
- uint8_t * pData
- uint8_t * pEnd

4.2.1 Detailed Description

Struct that will contain the received frame.

4.2.2 Member Data Documentation

4.2.2.1 uint8_t* FRAMER::pBegin

Points to the first character of the received frame

4.2.2.2 uint8_t* FRAMER::pData

Points to one position after the last received frame. So the number of received bytes = pData - pBegin

4.2.2.3 uint8_t* FRAMER::pEnd

Points to the last position of the buffer. This variable defines the maximum size of the buffer.

The documentation for this struct was generated from the following file:

· define.h

NeoRS232_AVR File Documentation

5.1 CRC_Test.c File Reference

#include "define.h"

Functions

- uint16_t get_crc (uint8_t *buf, uint8_t length)
- BOOL stripCRC (FRAMER *pFramer)
- void appendCRC (uint8_t *buffer, int length)

5.1.1 Detailed Description

The protocol makes use of checksums, these functions implement these.

5.1.2 Function Documentation

5.1.2.1 void appendCRC (uint8_t * buffer, int length)

Calculate a CRC16 of a buffer and append it to that buffer.

Parameters:

buffer databuffer of which CRC16 will be calculated and to which CRC16 will be appended. *length* number of bytes in the buffer

Returns:

the same buffer, with CRC-appended (little endian)

5.1.2.2 uint16_t get_crc (uint8_t * buf, uint8_t length)

Get the CRC16 of a databuffer.

Parameters:

buf containing data of which the CRC will be calculated

length total length of the buffer

Returns:

CRC16

5.1.2.3 BOOL stripCRC (FRAMER * pFramer)

check and strip the CRC16 of the frame.

Parameters:

pFramer A frame containing the received data.

Returns:

TRUE when correct CRC, else return FALSE, when the CRC is correct, then the CRC will be removed from the frame.

5.2 define.h File Reference

```
#include <avr/io.h>
#include <avr/crc16.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
```

Defines

```
#define _DEFINE_H_ 1
#define FOSC 16000000
#define FRAMESIZE 100

USART receive buffer size.
```

Enumerations

```
• enum FLAGS {
 LEADING_FLAG = 0xAA,
 DLE_FLAG = 0x10,
 TRAILING_FLAG = 0xFF,
 ACK0_FLAG = 0x30 }
    {\it USART\ protocol\ constants}.
• enum FRAME_TYPE {
 INFO_FRAME = 0,
 ACK_FRAME = 1 }
    The types of frames that can occur in the protocol.
• enum BOOL {
 FALSE,
 TRUE }
    Implementation of boolean type.
• enum MSG_T {
 MSG_NO_INPUT,
 MSG_DECODING,
 MSG_RX_COMPLETE }
```

States of the receiving state machine.

Functions

- void initTimer2 (void)
- void stopTimer2 (void)
- void writeString (uint8_t *buffer, uint8_t length, FRAME_TYPE typeFrame)
- MSG_T FramerReceive (FRAMER *pFr)
- void writeFrame (uint8 t *buffer, uint8 t length)
- BOOL stripCRC (FRAMER *pFramer)
- void appendCRC (uint8_t *buffer, int length)
- void init_USART (uint32_t baud, BOOL RX_ON)
- void write_USART (uint8_t *buf, uint8_t length)
- void write_USART_B (uint8_t c)
- BOOL read USART (uint8 t*c)
- void receiveInit (FRAMER *pFr)
- void receiveRun (FRAMER *pFramer)

5.2.1 Detailed Description

5.2.2 Define Documentation

5.2.2.1 #define _DEFINE_H_ 1

<Don't include this file twice

5.2.2.2 #define FOSC 16000000

MCU operating frequency

5.2.3 Enumeration Type Documentation

5.2.3.1 enum **BOOL**

Implementation of boolean type.

Enumeration values:

FALSE False, 0 **TRUE** True, 1

5.2.3.2 enum **FLAGS**

USART protocol constants.

Enumeration values:

LEADING_FLAG Start flag

DLE_FLAG Bytestuffing flag

TRAILING_FLAG End flag

ACK0_FLAG Acknowledge flag

5.2.3.3 enum FRAME_TYPE

The types of frames that can occur in the protocol.

Enumeration values:

```
INFO_FRAME Info frame (contains data)ACK_FRAME Acknowledge frame
```

5.2.3.4 enum **MSG_T**

States of the receiving state machine.

Enumeration values:

```
MSG_NO_INPUT No data has come in.MSG_DECODING State machine is decoding data.MSG_RX_COMPLETE Complete packet received.
```

5.2.4 Function Documentation

5.2.4.1 void appendCRC (uint8_t * buffer, int length)

Calculate a CRC16 of a buffer and append it to that buffer.

Parameters:

```
buffer databuffer of which CRC16 will be calculated and to which CRC16 will be appended. length number of bytes in the buffer
```

Returns:

the same buffer, with CRC-appended (little endian)

5.2.4.2 MSG_T FramerReceive (FRAMER * pFr)

Function that reads the received characters from the circular buffer using a state machine. The databytes are read from the circular buffer, a check if performed if it is a flagbyte. If it is, then the correct action is undertaken. If it is a databyte then that databyte is added to the framebuffer. A value is returned that tells us what this function has done with the data (if there was any).

Parameters:

pFr Framer to which the data will be posted.

Returns:

state of the state machine.

5.2.4.3 void init_USART (uint32_t baud, BOOL RX_ON)

Set the USART with baud, enable RX and TX, enable IRQ on receiving data.

Parameters:

```
baud Baudrate in [bps]
```

RX_ON Enable receiving UART-data when true.

5.2.4.4 void initTimer2 (void)

Initialize timer2, so that it will generate an interrupt after 5s.

5.2.4.5 **BOOL** read_USART (uint8 $_{t}*c$)

Function that reads a character from the circular receive buffer.

Parameters:

c Pointer to a character. The read character will be written to that address.

5.2.4.6 void receiveInit (FRAMER *pFr)

Initialize the frame so that it is ready to receive data

Parameters:

pFr Pointer to the frame that will be initialized.

5.2.4.7 void receiveRun (FRAMER * pFramer)

Receive run routine: checks if data has come in and act accordingly.

Parameters:

pFramer Check if data has come in on this framer

5.2.4.8 void stopTimer2 (void)

Stop timer2 by disconnecting clock source and disabling interrupts on this timer.

5.2.4.9 BOOL stripCRC (**FRAMER** * *pFramer*)

check and strip the CRC16 of the frame.

Parameters:

pFramer A frame containing the received data.

Returns:

TRUE when correct CRC, else return FALSE, when the CRC is correct, then the CRC will be removed from the frame.

5.2.4.10 void write_USART (uint8_t * buf, uint8_t length)

Send a characterbuffer to the USART

Parameters:

buf Pointer to buffer that contains data to be sent

length Number of bytes to be sent

5.2.4.11 void write_USART_B (**uint8_t** *c*)

Write one character to the UART

Parameters:

c The character that must be sent.

5.2.4.12 void writeFrame (uint8_t * buffer, uint8_t length)

Function that sends a frame. First a CRC16 is appended. Then the frame is sent. After that, a timeout timer is initialized. The acknowledge must come in before the timeout expires.

Parameters:

buffer Buffer containing the data to send.

length Number of bytes that must be sent.

5.2.4.13 void writeString (uint8_t * buffer, uint8_t length, FRAME_TYPE typeFrame)

Write a databuffer to the serial port. No CRC will be appended. This function adds the flags for the protocol and performs the bytestuffing. Finally it sends the data to the UART.

Parameters:

buffer Buffer containing the data.

length Length of the databuffer

typeFrame Type of frame (info frame or acknowledge frame)

5.3 main.c File Reference

```
#include "define.h"
```

Functions

- int main (void)

 Main function.
- void receiveInit (FRAMER *pFr)
- void receiveRun (FRAMER *pFramer)

Variables

- FRAMER framer
- uint8_t frameData [FRAMESIZE]

5.3.1 Detailed Description

The files in this project define a RS232-protocol.

5.3.2 Function Documentation

5.3.2.1 void receiveInit (FRAMER *pFr)

Initialize the frame so that it is ready to receive data

Parameters:

pFr Pointer to the frame that will be initialized.

5.3.2.2 void receiveRun (**FRAMER** * *pFramer*)

Receive run routine: checks if data has come in and act accordingly.

Parameters:

pFramer Check if data has come in on this framer

5.3.3 Variable Documentation

5.3.3.1 uint8_t frameData[FRAMESIZE]

Buffer that will contain the incoming frames

5.3.3.2 FRAMER framer

The framer contains pointers to work with the incoming data.

5.4 serieData.c File Reference

#include "define.h"

Functions

• void writeFrame (uint8_t *buffer, uint8_t length)

5.4.1 Detailed Description

Higher OSI-level function for sending frames.

5.4.2 Function Documentation

5.4.2.1 void writeFrame (uint8_t * buffer, uint8_t length)

Function that sends a frame. First a CRC16 is appended. Then the frame is sent. After that, a timeout timer is initialized. The acknowledge must come in before the timeout expires.

Parameters:

buffer Buffer containing the data to send.

length Number of bytes that must be sent.

5.5 simpleRead.c File Reference

#include "define.h"

Functions

• MSG_T FramerReceive (FRAMER *pFr)

5.5.1 Detailed Description

Functions implementing the protocol: reading data

5.5.2 Function Documentation

5.5.2.1 MSG_T FramerReceive (FRAMER * pFr)

Function that reads the received characters from the circular buffer using a state machine. The databytes are read from the circular buffer, a check if performed if it is a flagbyte. If it is, then the correct action is undertaken. If it is a databyte then that databyte is added to the framebuffer. A value is returned that tells us what this function has done with the data (if there was any).

Parameters:

pFr Framer to which the data will be posted.

Returns:

state of the state machine.

5.6 simpleWrite.c File Reference

#include "define.h"

Functions

- BOOL flagCharacterInData (uint8_t c)
- void writeString (uint8_t *buffer, uint8_t length, FRAME_TYPE typeFrame)

5.6.1 Detailed Description

Functions implementing the protocol: writing data

5.6.2 Function Documentation

5.6.2.1 BOOL flagCharacterInData (uint8_t c)

Check if the character is a flag character.

Parameters:

c The character

Returns:

TRUE when c is a flag character else FALSE.

5.6.2.2 void writeString (uint8_t * buffer, uint8_t length, FRAME_TYPE typeFrame)

Write a databuffer to the serial port. No CRC will be appended. This function adds the flags for the protocol and performs the bytestuffing. Finally it sends the data to the UART.

Parameters:

buffer Buffer containing the data.

length Length of the databuffer

typeFrame Type of frame (info frame or acknowledge frame)

5.7 timer.c File Reference

#include "define.h"

Functions

- INTERRUPT (SIG_OUTPUT_COMPARE2)
- void initTimer2 (void)
- void stopTimer2 (void)

5.7.1 Detailed Description

Timer functions used for controlling the time-out of the communication: You still have to fill in what to do when the message isn't sent correctly (e.g. send the frame again, generate an error, ...).

5.7.2 Function Documentation

5.7.2.1 void initTimer2 (void)

Initialize timer2, so that it will generate an interrupt after 5s.

5.7.2.2 INTERRUPT (SIG_OUTPUT_COMPARE2)

ISR of Timer2, will be called every 5s when timer2 is enabled.

5.7.2.3 void stopTimer2 (void)

Stop timer2 by disconnecting clock source and disabling interrupts on this timer.

5.8 usart.c File Reference 23

5.8 usart.c File Reference

```
#include "define.h"
```

Defines

• #define RXBUFSIZE 100

Functions

- void init_USART (uint32_t baud, BOOL RX_ON)
- void write_USART (uint8_t *buf, uint8_t length)
- void write_USART_B (uint8_t c)
- BOOL read_USART (uint8_t *c)
- SIGNAL (SIG_UART_RECV)

Variables

• uint8_t circRXArray [RXBUFSIZE]

5.8.1 Detailed Description

This function takes care of the USART. This is the lowest possible OSI-layer on the MCU.

5.8.2 Define Documentation

5.8.2.1 #define RXBUFSIZE 100

<Size of the circular buffer for receiving characters.

5.8.3 Function Documentation

5.8.3.1 void init_USART (uint32_t baud, BOOL RX_ON)

Set the USART with baud, enable RX and TX, enable IRQ on receiving data.

Parameters:

baud Baudrate in [bps]

RX_ON Enable receiving UART-data when true.

5.8.3.2 **BOOL** read_USART (uint8 $_{t}*c$)

Function that reads a character from the circular receive buffer.

Parameters:

c Pointer to a character. The read character will be written to that address.

5.8.3.3 SIGNAL (SIG_UART_RECV)

ISR-UART receive. This ISR will be called every time a character comes in. The character will then be written to a receive buffer.

5.8.3.4 void write_USART (uint8_t * buf, uint8_t length)

Send a characterbuffer to the USART

Parameters:

buf Pointer to buffer that contains data to be sentlength Number of bytes to be sent

5.8.3.5 void write_USART_B (uint8_t c)

Write one character to the UART

Parameters:

c The character that must be sent.

5.8.4 Variable Documentation

5.8.4.1 uint8_t circRXArray[RXBUFSIZE]

Circular array containing the incoming data bytes.

Index

_DEFINE_H_	MSG_T, 15
define.h, 14	read_USART, 16
	receiveInit, 16
ACK0_FLAG	receiveRun, 16
define.h, 14	stopTimer2, 16
ACK_FRAME	stripCRC, 16
define.h, 15	TRAILING_FLAG, 14
appendCRC	TRUE, 14
CRC_Test.c, 11	write_USART, 16
define.h, 15	write_USART_B, 16
	writeFrame, 17
BOOL	writeString, 17
define.h, 14	DLE_FLAG
. 1774	define.h, 14
circRXArray	deme.n, 14
usart.c, 24	FALSE
COM_HANDLE_T, 7	define.h, 14
pRxBegin, 7	flagCharacterInData
pRxEnd, 7	_
pRxGet, 7	simpleWrite.c, 21 FLAGS
pRxPut, 7	
rxCount, 7	define.h, 14
CRC_Test.c, 11	FOSC
appendCRC, 11	define.h, 14
get_crc, 11	FRAME_TYPE
stripCRC, 12	define.h, 14
1.6 1.10	frameData
define.h, 13	main.c, 18
_DEFINE_H_, 14	FRAMER, 9
ACK0_FLAG, 14	pBegin, 9
ACK_FRAME, 15	pData, 9
appendCRC, 15	pEnd, 9
BOOL, 14	framer
DLE_FLAG, 14	main.c, 18
FALSE, 14	FramerReceive
FLAGS, 14	define.h, 15
FOSC, 14	simpleRead.c, 20
FRAME_TYPE, 14	
FramerReceive, 15	get_crc
INFO_FRAME, 15	CRC_Test.c, 11
init_USART, 15	
initTimer2, 15	INFO_FRAME
LEADING_FLAG, 14	define.h, 15
MSG_DECODING, 15	init_USART
MSG_NO_INPUT, 15	define.h, 15
MSG_RX_COMPLETE, 15	usart.c, 23

26 INDEX

initTimer2	CICNAI
define.h, 15	SIGNAL
timer.c, 22	usart.c, 23 simpleRead.c, 20
INTERRUPT	simpleRead.c, 20
	FramerReceive, 20
timer.c, 22	simpleWrite.c, 21
LEADING_FLAG	-
define.h, 14	simpleWrite.c
define.ii, 11	flagCharacterInData, 21
main.c, 18	writeString, 21
frameData, 18	stopTimer2
framer, 18	define.h, 16
receiveInit, 18	timer.c, 22
receiveRun, 18	stripCRC
MSG_DECODING	CRC_Test.c, 12
define.h, 15	define.h, 16
MSG_NO_INPUT	timer.c, 22
define.h, 15	initTimer2, 22
MSG_RX_COMPLETE	INTERRUPT, 22
define.h, 15	stopTimer2, 22
MSG T	TRAILING_FLAG
define.h, 15	define.h, 14
	TRUE
pBegin	define.h, 14
FRAMER, 9	define.ii, 11
pData	usart.c, 23
FRAMER, 9	circRXArray, 24
pEnd	init_USART, 23
FRAMER, 9	read_USART, 23
pRxBegin	RXBUFSIZE, 23
COM_HANDLE_T, 7	SIGNAL, 23
pRxEnd	write_USART, 24
COM_HANDLE_T, 7	write_USART_B, 24
pRxGet	,
COM_HANDLE_T, 7	write_USART
pRxPut	define.h, 16
COM_HANDLE_T, 7	usart.c, 24
1 LICADE	write_USART_B
read_USART	define.h, 16
define.h, 16	usart.c, 24
usart.c, 23	writeFrame
receiveInit	define.h, 17
define.h, 16 main.c, 18	serieData.c, 19
receiveRun	writeString
define.h, 16	define.h, 17
main.c, 18	simpleWrite.c, 21
RXBUFSIZE	
usart.c, 23	
rxCount COM_HANDLE_T, 7	
COM_HANDLE_1, /	
serieData.c, 19	
serieData.c	
writeFrame, 19	
,	