USART Functions

TABLE OF CONTENTS

1 Introduction

The following routines are provided for devices with a single USART peripheral:

TABLE: SINGLE USART PERIPHERAL FUNCTIONS

Function	Description
BusyUSART	Is the USART transmitting?
CloseUSART	Disable the USART.
DataRdyUSART	Is data available in the USART read buffer?
getcUSART	Read a byte from the USART.
getsUSART	Read a string from the USART.
OpenUSART	Configure the USART.
putcUSART	Write a byte to the USART.
putsUSART	Write a string from data memory to the USART.
putrsUSART	Write a string from program memory to the USART.
ReadUSART	Read a byte from the USART.
WriteUSART	Write a byte to the USART.
baudUSART	Set the baud rate configuration bits for enhanced USART.

The following routines are provided for devices with multiple USART peripherals:

TABLE: MULTIPLE USART PERIPHERAL FUNCTIONS

Function	Description
BusyxUSART	Is USART x transmitting?
ClosexUSART	Disable USART x.
DataRdyxUSART	Is data available in the read buffer of USART x?
getcxUSART	Read a byte from USART x.
getsxUSART	Read a string from USART x.
OpenxUSART	Configure USART x.
putcxUSART	Write a byte to USART x.
putsxUSART	Write a string from data memory to USART x.
putrsxUSART	Write a string from program memory to USART x.
ReadxUSART	Read a byte from USART x.
WritexUSART	Write a byte to USART x.
baudxUSART	Set the baud rate configuration bits for enhanced USART x.

Based on the different control registers, configuration bits and their positions in the control register, all PIC18 devices are divided into following different versions. Wherever required, separate functions have been designed to support these versions, so before calling the LIB functions care has to be taken to know the version of the configured device and to call the appropriate function with correct number of arguments. For EAUSART_V11 user has to configure the I/O registers.
Below is the table to find the USART version for the configured device:

TABLE: VERSION VS. DEVICES

Version name	Device number

AUSART_V1	18C242, 18C252, 18C442, 18C452, 18F242, 18F252, 18F442, 18F452, 18F248, 18F258, 18F448, 18F458, 18F2439, 18F2539, 18F4439, 18F4539, 18C601, 18C801, 18C658, 18C858, 18F2220, 18F2320, 18F4220, 18F4320
AUSART_V2	18F6620, 18F6720, 18F8620, 18F8720, 18F6520, 18F8520
EAUSART_V3	18F1220, 18F1320, 18F6585, 18F6680, 18F8585, 18F8680, 18F2331, 18F2431, 18F4331, 18F4431
EAUSART_V4	18F1230, 18F1330, 18F1231, 18F1331, 18F2420, 18F2520, 18F4420, 18F4520, 18F2423, 18F2523, 18F4423, 18F4523, 18F2450, 18F4450, 18F2480, 18F2580, 18F4480, 18F4580, 18F2410, 18F2510, 18F2515, 18F2610, 18F4410, 18F4510, 18F4515, 18F4610, 18F2525, 18F2620, 18F4525, 18F4620, 18F2585, 18F2680, 18F4585, 18F4680, 18F2682, 18F2685, 18F4682, 18F4685, 18F24J10, 18F25J10, 18F44J10, 18F4510
EAUSART_V5	18F2455, 18F2550, 18F4455, 18F4550, 18F2221, 18F2321, 18F4221, 18F4321, 18F23K20, 18F24K20, 18F25K20, 18F43K20, 18F44K20, 18F45K20, 18F13K50, 18LF13K50, 18F14K50, 18LF14K50, 18F13K22, 18F14K22, 18LF13K22, 18LF14K22
EAUSART_V6	18F6310, 18F6410, 18F8310, 18F8410, 18F6390, 18F6490, 18F8390, 18F8490, 18F63J11, 18F64J11, 18F65J11, 18F8311, 18F84J11, 18F85J11, 18F63J90, 18F64J90, 18F65J90, 18F83J90, 18F84J90, 18F85J90, 18F66J90, 18F67J90, 18F86J90, 18F87J90
EAUSART_V7	18F6527, 18F6622, 18F6627, 18F6722, 18F8527, 18F8622, 18F8627, 18F8722, 18F65J10, 18F65J15, 18F66J10, 18F6615, 18F67J10, 18F85J10, 18F85J15, 18F86J10, 18F86J15, 18F87J10
EAUSART_V8	18F6525, 18F6621, 18F8525, 18F8621
EAUSART_V9	18F86J60, 18F86J65, 18F87J60, 18F96J60, 18F96J65, 18F7J60, 18F66J11, 18F66J16, 18F67J11, 18F86J11, 18F86J1, , 18F87J11, 18F65J50, 18F66J50, 18F66J55, 18F67J50, 18F, 5J50, 18F86J50, 18F86J55, 18F87J50
EAUSART_V10	18F66J60, 18F66J65, 18F67J60
EAUSART_V11	18F24J11, 18F25J11, 18F26J11, 18F44J11, 18F45J11, 18F46J11, 18F24J50, 18F25J50, 18F26J50, 18F44J50, 18F45J50, 18F46J50, 18LF24J11, 18LF25J11, 18LF26J11, 18LF44J11, 18LF45J11, 18LF46J11, 18LF24J50, 18LF25J50, 18LF26J50, 18LF44J50, 18LF45J50, 18LF46J50

2 Function Descriptions

2.1 BusyUSART
Busy1USART
Busy2USART

Function: Is the USART transmitting?

Include: `usart.h`

Prototype:
```
char BusyUSART(  void );
char Busy1USART( void );
char Busy2USART( void );
```

Remarks: Returns a value indicating if the USART transmitter is currently busy. This function should be used prior to commencing a new transmission.

`BusyUSART` should be used on parts with a single USART peripheral. `Busy1USART` and `Busy2USART` should be used on parts with multiple USART peripherals.

Return Value: `0` if the USART transmitter is idle

`1` if the USART transmitter is in use

File Name: `ubusy.c`
`u1busy.c`
`u2busy.c`

Code Example: while (BusyUSART());

2.2 CloseUSART Close1USART Close2USART

Function: Disable the specified USART.

Include: `usart.h`

Prototype:
```
void CloseUSART(  void );
void Close1USART( void );
void Close2USART( void );
```

Remarks: This function disables the interrupts, transmitter and receiver for the specified USART.

`CloseUSART` should be used on parts with a single USART peripheral. `Close1USART` and `Close2USART` should be used on parts with multiple USART peripherals.

File Name: `uclose.c`
`u1close.c`
`u2close.c`

2.3 DataRdyUSART DataRdy1USART DataRdy2USART

Function: Is data available in the read buffer?

Include: `usart.h`

Prototype:
```
char DataRdyUSART(  void );
char DataRdy1USART( void );
char DataRdy2USART( void );
```

Remarks: This function returns the status of the RCIF flag bit in the PIR register.

`DataRdyUSART` should be used on parts with a single USART peripheral. `DataRdy1USART` and `DataRdy2USART` should be used on parts with multiple USART peripherals.

Return Value: `1` if data is available
`0` if data is not available

File Name: `udrdy.c`
`u1drdy.c`
`u2drdy.c`

Code Example:
```
while (!DataRdyUSART());
```

2.4 getcUSART getc1USART getc2USART

`getcxUSART` is defined as `ReadxUSART`. See `ReadUSART`

2.5 getsUSART gets1USART gets2USART

Function: Read a fixed-length string of characters from the specified USART.

Include: `usart.h`

Prototype:
```
void getsUSART ( char * buffer,
                 unsigned char len );
void gets1USART ( char * buffer,
                  unsigned char len );
void gets2USART ( char * buffer,
```

```
                                unsigned char len );
```

Arguments: ***buffer***

A pointer to the location where incoming characters are to be stored.

len

The number of characters to read from the USART.

Remarks: This function only works in 8-bit transmit/receive mode. This function waits for and reads len number of characters out of the specified USART. There is no time out when waiting for characters to arrive.

`getsUSART` should be used on parts with a single USART peripheral. `gets1USART` and `gets2USART` should be used on parts with multiple USART peripherals.

File Name: `ugets.c`
`u1gets.c`
`u2gets.c`

Code Example:
```
char inputstr[10];
getsUSART( inputstr, 5 );
```

2.6 OpenUSART Open1USART Open2USART

Function: Configure the specified USART module.

Include: `usart.h`

Prototype:
```
void OpenUSART( unsigned char config,
          unsigned int spbrg);
void Open1USART( unsigned char config,
           unsigned int spbrg);
void Open2USART( unsigned char config,
           unsigned int spbrg);
```

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('&') or a bitwise OR ('|'), configurable either way as shown on the example at the end of this file, with a value from each of the categories listed below. These values are defined in the file usart.h.

Interrupt on Transmission:

`USART_TX_INT_ON`	Transmit interrupt ON
`USART_TX_INT_OFF`	Transmit interrupt OFF

Interrupt on Receipt:

`USART_RX_INT_ON`	Receive interrupt ON
`USART_RX_INT_OFF`	Receive interrupt OFF

USART Mode:

`USART_ASYNCH_MODE`	Asynchronous Mode
`USART_SYNCH_MODE`	Synchronous Mode

Transmission Width:

`USART_EIGHT_BIT`	8-bit transmit/receive
`USART_NINE_BIT`	9-bit transmit/receive

Slave/Master Select*:

`USART_SYNC_SLAVE`	Synchronous Slave mode
`USART_SYNC_MASTER`	Synchronous Master mode

Reception mode:

`USART_SINGLE_RX`	Single reception
`USART_CONT_RX`	Continuous reception

Baud rate:

USART_BRGH_HIGH	High baud rate
USART_BRGH_LOW	Low baud rate

* Applies to Synchronous mode only

spbrg

This is the value that is written to the baud rate generator register which determines the baud rate at which the USART operates. The formulas for baud rate are:

Asynchronous mode, high speed:

Fosc / (16 * (spbrg + 1))

Asynchronous mode, low speed:

Fosc / (64 * (spbrg + 1))

Synchronous mode:

Fosc / (4 * (spbrg + 1))

Where Fosc is the oscillator frequency.

Remarks: This function configures the USART module according to the specified configuration options.

OpenUSART should be used on parts with a single USART peripheral. Open1USART and Open2USART should be used on parts with multiple USART peripherals.

File Name: uopen.c
u1open.c
u2open.c

Code Example:

```
OpenUSART1( USART_TX_INT_OFF  &
            USART_RX_INT_OFF  &
            USART_ASYNCH_MODE &
            USART_EIGHT_BIT   &
            USART_CONT_RX     &
            USART_BRGH_HIGH,
            25                );
```

2.7 putcUSART putc1USART putc2USART

putcxUSART is defined as WritexUSART. See WriteUSART

2.8 putsUSART puts1USART puts2USART putrsUSART putrs1USART putrs2USART

Function: Writes a string of characters to the USART including the null character.

Include: usart.h

Prototype:

```
void putsUSART(  char *data );
void puts1USART( char *data );
void puts2USART( char *data );
void putrsUSART(  const rom char *data );
void putrs1USART( const rom char *data );
void putrs2USART( const rom char *data );
```

Arguments: ***data***

Pointer to a null-terminated string of data.

Remarks: This function only works in 8-bit transmit/receive mode. This function writes a string of data to the USART including the null character.

Strings located in data memory should be used with the “puts” versions of these functions.

Strings located in program memory, including string literals, should be used with the “putrs” versions of these functions.

`putsUSART` and `putrsUSART` should be used on parts with a single USART peripheral. The other functions should be used on parts with multiple USART peripherals.

File Name: `uputs.c`
`u1puts.c`
`u2puts.c`
`uputrs.c`
`u1putrs.c`
`u2putrs.c`

Code Example:
```
PutrsUSART (“Hello World!”);
```

2.9 ReadUSART Read1USART Read2USART getcUSART getc1USART getc2USART

Function: Read a byte (one character) out of the USART receive buffer, including the 9th bit if enabled.

Include: `usart.h`

Prototype:
```
char ReadUSART(  void );
char Read1USART( void );
char Read2USART( void );
char getcUSART(  void );
char getc1USART( void );
char getc2USART( void );
```

Remarks: This function reads a byte out of the USART receive buffer. The Status bits and the 9th data bits are saved in a union with the following declaration:

```
union USART
{
  unsigned char val;
  struct
  {
    unsigned RX_NINE:1;
    unsigned TX_NINE:1;
    unsigned FRAME_ERROR:1;
    unsigned OVERRUN_ERROR:1;
    unsigned fill:4;
  };
};
```

The 9th bit is read-only if 9-bit mode is enabled. The Status bits are always read.

On a part with a single USART peripheral, the `getcUSART` and `ReadUSART` functions should be used and the status information is read into a variable named `USART_Status` which is of the type USART described above.

On a part with multiple USART peripherals, the `getcxUSART` and `ReadxUSART` functions should be used and the status information is read

into a variable named `USARTx_Status` which is of the type USART described above.

Return Value: This function returns the next character in the USART receive buffer.

File Name:
```
uread.c
u1read.c
u2read.c
#define in usart.h
#define in usart.h
#define in usart.h
```

Code Example:
```
int result;
result = ReadUSART();
result |= (unsigned int)
          USART_Status.RX_NINE << 8;
```

2.10 WriteUSART
Write1USART
Write2USART
putcUSART
putc1USART
putc2USART

Function: Write a byte (one character) to the USART transmit buffer, including the 9th bit if enabled.

Include: `usart.h`

Prototype:
```
void WriteUSART(  char data );
void Write1USART( char data );
void Write2USART( char data );
void putcUSART(  char data );
void putc1USART( char data );
void putc2USART( char data );
```

Arguments:

data

The value to be written to the USART.

Remarks: This function writes a byte to the USART transmit buffer. If 9-bit mode is enabled, the 9th bit is written from the field TX_NINE, found in a variable of type USART:

```
union USART
{
  unsigned char val;
  struct
  {
    unsigned RX_NINE:1;
    unsigned TX_NINE:1;
    unsigned FRAME_ERROR:1;
    unsigned OVERRUN_ERROR:1;
    unsigned fill:4;
  };
};
```

On a part with a single USART peripheral, the `putcUSART` and `WriteUSART` functions should be used and the Status register is named `USART_Status` which is of the type USART described above.

On a part with multiple USART peripherals, the `putcxUSART` and `WritexUSART` functions should be used and the status register is named `USARTx_Status` which is of the type USART described above.

File Name: uwrite.c
u1write.c
u2write.c
#define in usart.h
#define in usart.h
#define in usart.h

Code Example:
```
unsigned int outval;
USART1_Status.TX_NINE = (outval &
0x0100)                    >> 8;
Write1USART( (char) outval );
```

2.11 baudUSART
baud1USART
baud2USART

Function: Set the baud rate configuration bits for enhanced USART operation.

Include: usart.h

Prototype:
```
void baudUSART(  unsigned char baudconfig );
void baud1USART( unsigned char baudconfig  );
void baud2USART( unsigned char baudconfig  );
```

Arguments: ***baudconfig***

A bitmask that is created by performing a bitwise AND ('&') operation with a value from each of the categories listed below. These values are defined in the file usart.h:

RX Idle State:
In Asynchronous mode:

BAUD_IDLE_RX_PIN_STATE_HIGH	Receive data (RX) is inverted.Idle state for RX pin is high level
BAUD_IDLE_RX_PIN_STATE_LOW	No inversion of receive data(RX). Idle state for RX pin is low level

In Synchronous mode:

BAUD_IDLE_RX_PIN_STATE_HIGH	Data (DT) is inverted. Idle state for DT pin is high level
BAUD_IDLE_RX_PIN_STATE_LOW	No inversion of data (DT). Idle state for DT pin is low level

TX Idle State (In Asynchronous mode):

BAUD_IDLE_TX_PIN_STATE_HIGH	Transmit data (TX) is inverted. Idle state for TX pin is high level
BAUD_IDLE_TX_PIN_STATE_LOW	No inversion of transmit data (TX). Idle state for TX pin is low level

Clock Idle State: (In Synchronous mode)

BAUD_IDLE_CLK_HIGH	Clock idle state is a high level
BAUD_IDLE_CLK_LOW	Clock idle state is a low level

Baud Rate Generation:

BAUD_16_BIT_RATE	16-bit baud generation rate
BAUD_8_BIT_RATE	8-bit baud generation rate

RX Pin Monitoring:

BAUD_WAKEUP_ON	RX pin monitored
BAUD_WAKEUP_OFF	RX pin not monitored

Baud Rate Measurement:

BAUD_AUTO_ON	Auto baud rate measurement enabled

BAUD_AUTO_OFF	Auto baud rate measurement disabled

Remarks: These functions are only available for processors with enhanced USART capability.

File Name: ubaud.c
u1baud.c
u2baud.c

Code Example:
```
baudUSART (BAUD_IDLE_CLK_HIGH &
           BAUD_16_BIT_RATE &
           BAUD_WAKEUP_ON &
           BAUD_AUTO_ON);
```

Example Use of the USART Routines with AND mask:

```
#include <p18C452.h>
#include <usart.h>

void main(void)
{
  // configure USART
  OpenUSART( USART_TX_INT_OFF  &
             USART_RX_INT_OFF  &
             USART_ASYNCH_MODE &
             USART_EIGHT_BIT   &
             USART_CONT_RX     &
             USART_BRGH_HIGH,25 );

  while(1)
  {
    while( ! PORTAbits.RA0 );  //wait for RA0 high

    WriteUSART( PORTD );       //write value of PORTD

    if(PORTD == 0x80)          // check for termination
      break;                   //   value
  }

  CloseUSART();
}
```

Example Use of the USART Routines with OR mask:

```
#include <p18C452.h>
#define USE_OR_MASKS
#include <usart.h>

void main(void)
{
  // configure USART
  OpenUSART( USART_TX_INT_OFF  |
             USART_RX_INT_OFF  |
             USART_ASYNCH_MODE |
             USART_EIGHT_BIT   |
             USART_CONT_RX     |
             USART_BRGH_HIGH,25 );

  while(1)
  {
    while( ! PORTAbits.RA0 );  //wait for RA0 high

    WriteUSART( PORTD );       //write value of PORTD

    if(PORTD == 0x80)          // check for termination
      break;                   //   value
  }
```

```
  CloseUSART();
}
```