AVR1307: Using the XMEGA USART

Features

- **Setup and use of the USART**
- **Code examples**
  - **Polled USART**
  - **Interrupt controlled USART**

1 Introduction

The USART (Universal Synchronous Asynchronous Receiver Transmitter) is the key element in serial communications between computers, terminals and other devices.

This application note describes how to set up and use the USART in asynchronous mode in the XMEGA™. C code drivers and examples are included for both polled and interrupt controlled USART applications. Polled version supports all character sizes (5-9 bit) and also serves as an example for how the interrupt based version can be modified to support 9-bit characters. The interrupt based driver supports only 5-8 bit characters.

This application note covers the asynchronous mode, in which the transmitter and receiver clock are independent and not synchronized.

8-bit AVR® Microcontrollers

Application Note

Rev. 8049A-AVR-02/08

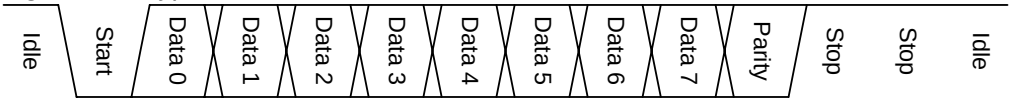
2 Theory of operation

UART communication is character based, and a character is defined to be 5, 6, 7, 8 or 9 bits. A character is preceded by a start bit, and followed by one or two stop bits. The start bit is always low, as opposed to the high IDLE state of the line.

It is optional to include a parity bit, which is placed after the data character, and before the stop bit(s). The parity bit offers detection of single bit faults, but is less efficient for multiple bit faults.

Figure 2-1 illustrates a standard data transfer, including 1 start bit, 8 data bits, 1 parity bit and 2 stop bits.

Figure 2-1. Typical data transmission.



2.1 The XMEGA USART module

The XMEGA USART is by default set in UART (asynchronous) mode. Transmit and receive are enabled individually, and the transfers can be implemented using polling of the corresponding flags or by interrupt routines. Except when using the 9-bit character size the use of the USART has few pitfalls.

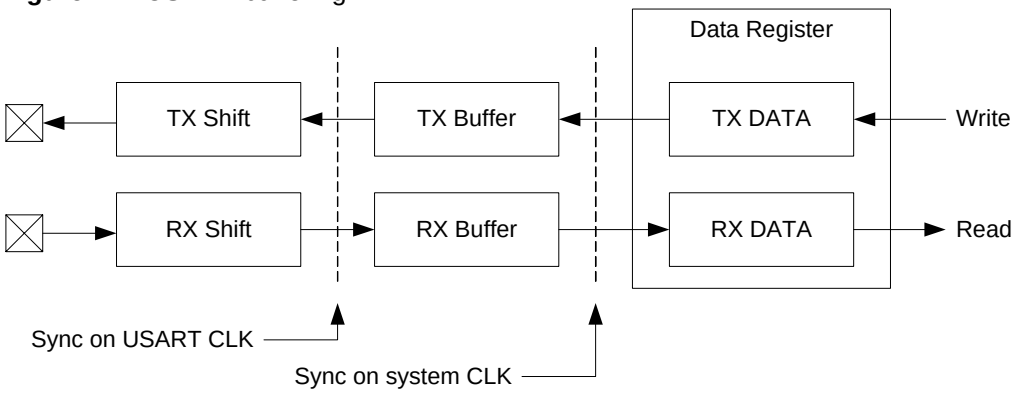
Special care must be taken if using 9-bit characters. For data reception, the 9th bit (RXB8) is located in the status register (USARTxn.STATUS), and must be included when reading the received data from the USART. As the RXCIF flag is cleared after reading the USARTxn.DATA register, RXB8 must be read before the least significant byte from in the data register (USARTxn.DATA).

For data transmission, the 9th bit (TXB8) is located in Control Register B (USARTxn.CTRLB). Writing the low byte to USARTxn.DATA will trigger the transmission, so TXB8 must be written first.

2.1.1 Buffers

The USART is double buffered in both receive (RX) and transmit (TX) direction. In addition to the RX and TX shift registers connected to the IO pins, there are two buffer registers for both RX and TX. One buffer is referred to as the DATA register, while the other is referred to as a buffer register. During normal operation one would only access the DATA register, as data is automatically moved by hardware between the DATA buffer and shift registers. Both RX and TX DATA registers share the same physical address, but a read will access the receive DATA register, and a write the transmit DATA register.

Figure 2-2. USART buffering.



When a complete character is received in the shift register, it is copied to the buffer, and the shift register is ready to receive a second (or third) character. Note that, if three characters have been received, without reading the DATA register, a fourth character will cause loss of the character in the shift register, the third byte is lost. In this case the buffer overflow flag will be set (BUFOVF in USARTxn.STATUS). The application must therefore read received data fast enough to avoid data loss, which may require additional software buffers or an interrupt based driver.

The transmit buffer is like the receive buffer a three-level FIFO buffer, and opens for a third character to be written as soon as first buffer level is copied to the shift register and transmission started.

2.1.2 Flags

The flags for Receive Complete (RXCIF), Transmit Complete (TXCIF) and Data Register Empty (DREIF), are essential in the USART operation.

The RXCIF flag is set when there are unread data in the receive buffer, and cleared when the receive buffer is empty. The RXCIF flag is cleared by reading the data, it is not required to clear the flag manually. TXCIF is set when an entire frame in the transmit shift register has been shifted out, and there are no new data currently present in the transmit buffer. The DREIF flag indicates that the transmit buffer (USARTxn.DATA) has room for additional data.

TXCIF and DREIF may seem similar, but there is a small difference that can be utilized to speed up data transfers. TXCIF is not set until the USART has completed transmitting all data in the transmit shift register and the transmit buffers are empty. DREIF on the other hand, is set already when the buffer has room for more data. This means new data can be filled into the buffer before all data is transmitted, and the small pauses between each transmitted characters can be avoided.

All three flags can be used to generate interrupts.

2.1.3 Baud rate selection

In addition to the baud rate clock generator controlled by the BSEL[11:0] bits in the USARTxn.BAUDCTRLA and -B registers, the XMEGA have the BSCALE[3:0] bits located in the high nibble of USARTxn.BAUDCTRLB, which controls the arithmetic baud rate clock setting.

If setting the BSCALE bits to 0 (default value), the baud rate generator operates without the extra scaling, and the baud rate is set by BSEL[11:0] according to Equation 2-1.

Equation 2-1. Baud rate for USART in Asynchronous Normal Speed mode.

$$f_{BAUD} = \frac{f_{PER}}{16(BSEL[11:0]+1)}$$

By using the standard mode alone, a wide selection of baud rates is achievable, but the BSCALE[3:0] bits add even more flexibility. Setting these bits to anything between –7 and 7 (except 0), will enable the arithmetic clock scaling, and extend the available baud rates. See the XMEGA manual for more information on the Baud rate scale factor and the double speed mode.

2.2 DMA transfers

The DMA functionality of XMEGA allows transfers of data to the USART. This can be used to reduce the CPU load when larger data blocks are to be transmitted/received. The DMA can be set up to generate interrupt when transfers of data blocks are completed.

For more information about the DMA, please refer to application note AVR1304.

3 USART drivers

This application note includes a source code package with a basic driver implemented in C. It is written in the IAR Embedded Workbench® compiler.

Note that this driver is written to be highly readable and as a general example how to use the peripheral module. Many of the function in the driver are implemented as macros to make the driver more speed and size efficient. In CPU intensive applications, the interrupt based driver shows an advantage. By using interrupt controlled driver, the CPU will not have to check if data are received or transmitted, but will automatically be notified when this occurs.

The choice of polled versus interrupt-driven drivers is application dependent, and often relies on protocol used for data transfers.

The code in the examples (polled and interrupt example) sends some values and checks that the values received are equal to the values sent. It can be tested, using a loop-back wire between I/O pins PIN2 and PIN3 on PORT C.

3.1 Files

The source code consists of three files:

- *usart_driver.c* – polling driver source file
- *usar_driver.h* – polling driver header file
- *usart_example_polled.c* – example code using the polling driver
- *usart_example_interrupt.c* – example code using the interrupt driver

3.2 9-bit character version

Only the polled driver is made to support 9 bits characters, but with some minor modifications the interrupt driver can support it too. (See Section 3.3) The polled driver includes special versions of USART_PutChar and USART_GetChar. These 9-bit versions are USART_NineBits_PutChar and USART_NineBits_GetChar, and are adapted to handle all 9 bits.

3.3 Interrupt driver

Some adaptations might be done to make sure the interrupt drivers fit the application.

3.3.1 Buffers

If support for 9-bit characters is needed in the interrupt-controlled driver, please see the polled driver for code examples. In addition to read and write the 9th bit, it's important to enlarge the buffer to hold 9-bit values as illustrated below.

```
typedef struct Buffer
{
  uint16_t RX[USART_RX_BUFFER_SIZE];
  uint16_t TX[USART_TX_BUFFER_SIZE];
  uint8_t RX_Head, RX_Tail;
  uint8_t TX_Head, TX_Tail;
} Buffer_t;
```

3.3.2 Interrupt service routines

The example is written using USART C0. If any other USART is to be used, remember to add interrupt service routines similar to those found for USART C0 below.

```
#pragma vector=USARTC0_RXC_vect
__interrupt void USARTC0_RxcIsr(void)
{
  USART_Rxc(&USART_C0);
}

#pragma vector=USARTC0_DRE_vect
__interrupt void USARTC0_DreIsr(void)
{
  USART_Dre(&USART_C0);
}
```

The interrupt service routines should call the common handlers (USART_Rxc and USART_Dre) with pointer to the correct USART as argument.

3.3.3 Ring buffer

Figure 3-1 illustrates the type of ring buffer used in the interrupt-controlled driver. A ring buffer is an array imagined to be circular. The instance writing to the buffer moves forward one step each time it writes, and when passing the end it will start again from the beginning. The instance reading moves the same way, and as long as data is read at the same or higher speed than written, the buffer acts like a queue. If data over time is written faster than read, the buffer will fill up.

Figure 3-1. Ring buffer with capacity of N elements.

3.4 Doxygen documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit http://www.doxygen.org. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the *readme.html* file in the source code folder.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature