AVR304: Half Duplex Interrupt Driven Software UART

8-bit AVR® Microcontrollers

Application Note

Features

- **Runs asynchronously (interrupt driven)**
- **Capable of handling baud rates of up to 38.4 kbaud @8MHz clock frequency**
- **Runs on any AVR device with 8-bit timer/counter and external interrupt**

1 Introduction

Lots of control applications communicate serially in one direction at a time only (half duplex). This application note describes how to make a half duplex UART on any AVR device using the 8-bit Timer/Counter0 and an external interrupt. This software can be used to implement a serial port on a device with no hardware UART, or it can be used to implement a second serial-port on AVR devices already equipped with a UART. Idle line is signaled by holding the line at logical one. The start bit is always a zero, and the UART receiver will detect the start of a frame by the first falling edge. Following the start bit come the data bits, followed by a stop bit which is always a logical one. This stop bit is held stable at one until the next start bit is sent.

Figure 1-1. UART Communication Frame Format

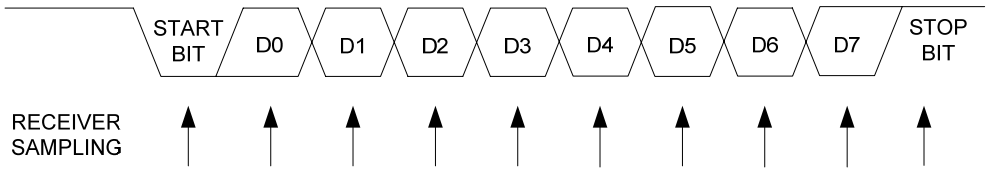
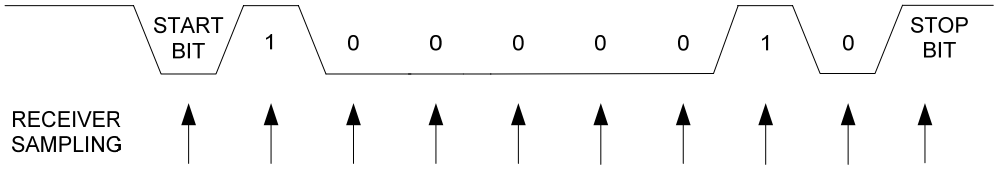


Figure 1-2. Serial frame of ASCII “A” ($41)



Rev. 0941C-AVR-04/08

2 Theory of Operation

Asynchronous serial data communication follows some simple rules on data transfer. Data is transmitted sequentially, one bit at a time. To inform the receiver that a new byte is arriving, each byte is placed between so-called start- and stop bits. This construction is called a frame. The frame format is shown in Figure 1-1 and Figure 1-2. The frame has 1 start-bit, 8 data bits, and 1 stop-bit. This frame type is implemented in this application note. The frame format can be extended, and might also include parity bits and more stop bits. In asynchronous transmissions, no separate clock is provided to the receiver. Correct reception of data is guaranteed by keeping all bit lengths equal. The receiver will synchronize from the first falling edge of the start bit, and find the next sampling time with its own timer. The bit length is determined by the baud rate used for the communication. In the case of a UART, the baud rate is equal to the number of bits transmitted per second. The transmitter and receiver have to be set up using the same baud rate equally for correct reception.

As seen in Figure 1-1, the frame starts with a falling edge. This falling edge generates an initial interrupt (using external interrupt) in the receiver. The interrupt starts Timer/Counter0, which is set to time out in exactly 1.5 bit lengths. This 1.5 bit length delay is required to generate the next sampling event in the middle of the first data bit. The next 8 interrupts are generated after a predefined delay (1 bit length) using counter/timer 0. Figure 2-1 shows the flowchart for receiving serial data.

Transmitting data is even easier, as all bits have equal length and the timer can be preset at a constant delay (1 bit-length). The first bit is the start-bit. This is always a logical zero (or space). This bit informs the receiver that data is coming. Then the data bits can be shifted out, LSB first (least significant bit) first, MSB last. Finally, the last bit must be a stop bit so the receiver can separate the data bytes. This is always a logical one (or mark). Figure 2-2 shows the flowchart for transmitting serial data.

Figure 2-1. Flowchart for receiving serial data.

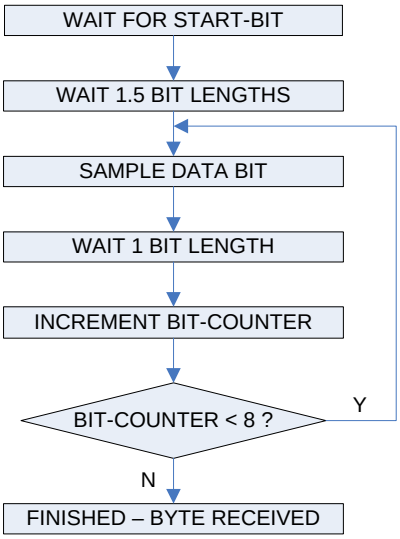
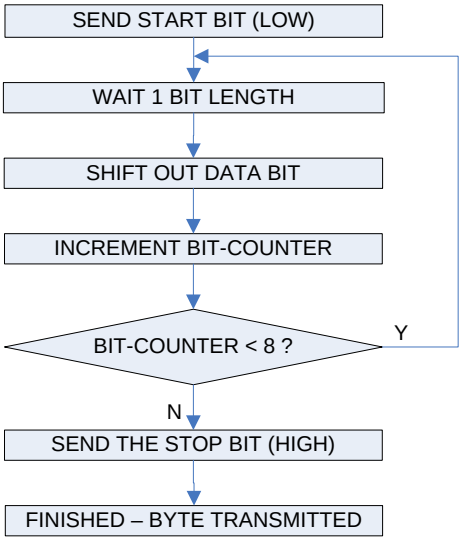


Figure 2-2. Flowchart for transmitting serial data.

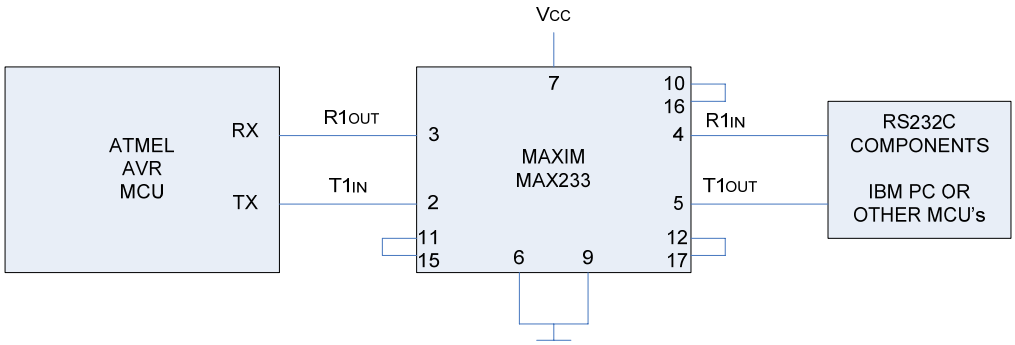


3 Connection

The RS-232 standard requires a voltage level of -15/+15V. To generate these signaling levels, a separate interface circuit is needed which converts the MCU's voltage to the RS-232 voltage. An example of a single chip interface circuit is MAXIM's MAX233. It operates from a single 5V power supply, and has an onchip DC-DC converter to convert the 5 volts to the RS-232 signaling levels.

The receive pin (RX) must be connected to the INT0-pin because of the external interrupt. It is not important which pin is used as the transmit pin (TX). Figure 3-1 shows how the MCU should physically be connected to an RS-232 line.

Figure 3-1. Physical connection to an RS232 serial line.



4 Implementation

These software UART routines use Timer/Counter 0 and one external interrupt. The clock provided to the MCU will limit the maximum baud rate obtainable. This software UART is capable of handling baud rates up to 38.400 kbit/s, at 8MHz clock frequency. At this speed nearly all computing power is used, but the MCU is still available for other tasks between each byte being transmitted.

The bit length is determined by the number of cycles (C•N) required to generate another overflow. With the Timer/Counter. N is the value loaded in the timer/counter compare register, and C is the Timer/Counter 0 prescaling factor, as described in the T/C Prescaler in the AVR datasheet. The value N can be calculated from the following formula, where Xtal is the frequency of the system:

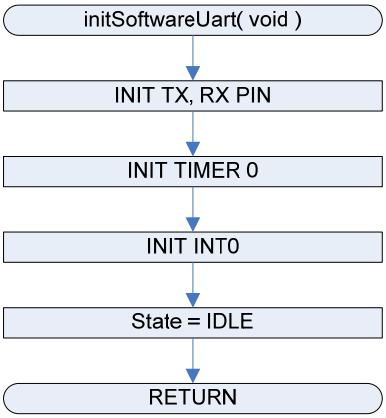
$$N = \frac{Xtal}{BaudRate \cdot C}$$

Note that the prescaling factor C should be one of the values 1, 8, 64, 256, or 1024.

5 “initSoftwareUart( )” FUNCTION – INITIALIZE UART

Before data can be transferred using the UART, the UART has to be initialized by calling the function “initSoftwareUart()”. This function will set up the Timer/Counter prescaler, and enable the Timer/counter and external interrupt needed for communication. Upon return from the subroutine, a '__enable_interrupt()' instruction should follow to enable global interrupts. This will enable the UART. By issuing a '__disable_interrupt()' instruction at a later time, the UART can be disabled.

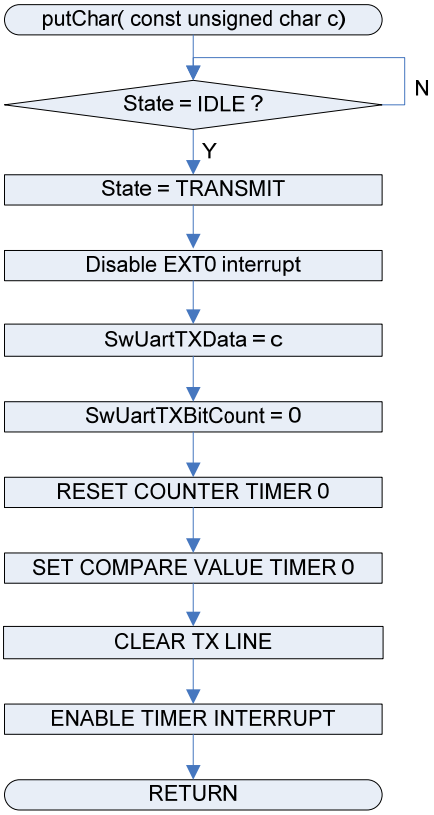
Figure 5-1. “initSoftwareUart( )” Flow Chart.



6 “putChar(const unsigned char)” FUNCTION - TRANSMITTING A BYTE

This routine is used when the software needs to transmit data. It waits for the IDLE state, then sets it to TRANSMIT, disables the external interrupt (to disable reception when transmitting), sets the correct baud rate (t/c0 interrupt) and outputs the start bit.

Figure 6-1. “putChar” Flow Chart.



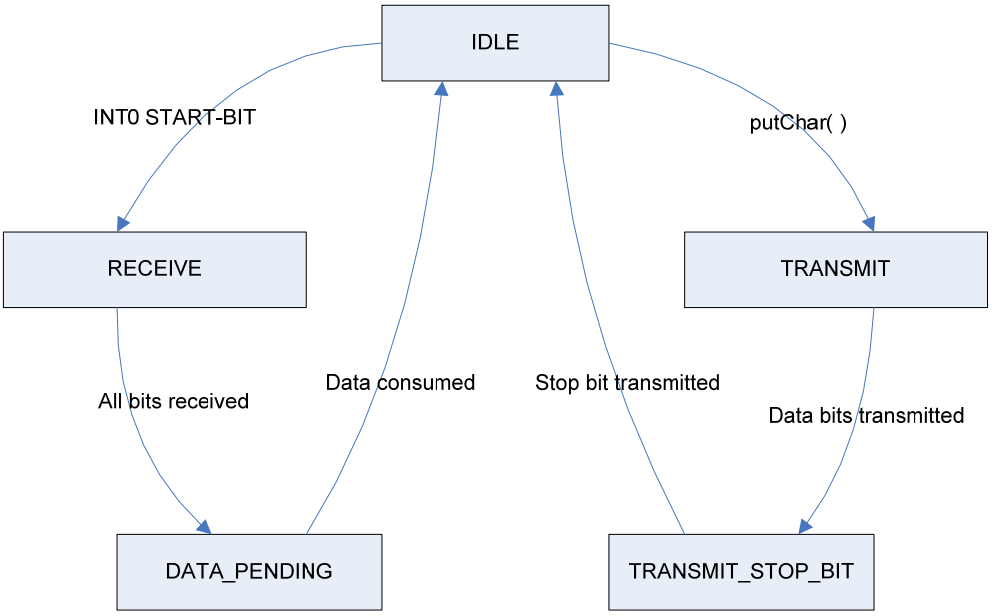
7 “state” enumerate

The state enum is used by all functions. It has five states implemented:

1. IDLE: the system is in idle state and both receiving and transmitting is possible.
2. TRANSMIT: transmitting bits. INT0 interrupts are disabled, system can not receive.
3. TRANSMIT_STOP_BIT: state for stop bit. This is to ensure at least one stop bit is transmitted. After one stop bit is transmitted, the system state is set to IDLE.
4. RECEIVE: receiving bits. When interrupted on INT0, the system changes state to RECEIVE, and receives 8 bits before changing state to DATA_PENDING. No sending can occur at the same time.
5. DATA_PENDING: when done receiving. Received data is stored in SwUartRXData and are ready to be processed. After data is processed the state should be set to IDLE. This state does not stop new receptions from occurring, so the data may be lost if not processed right away.

State transition is shown in Figure 7-1.

Figure 7-1. State transition diagram.



8 “Timer0_Compare_interrupt( )” INTERRUPT SERVICE ROUTINE

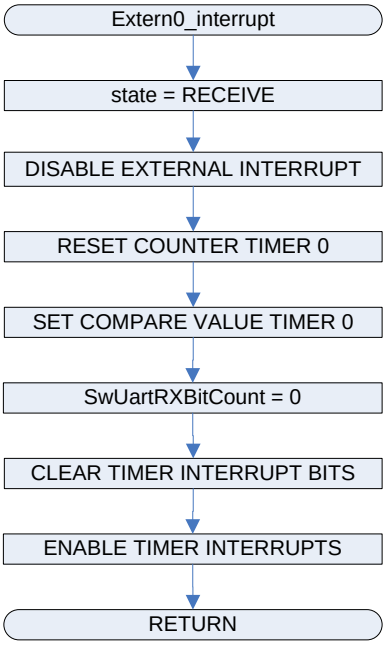
This routine takes care of sending and receiving each bit in the transmission. The routine is called automatically on Timer/Counter compare match, to send or receive the next bit.

The Timer/Counter overflow interrupt is enabled by “putChar” or “Extern0_interrupt”, when transmitting or receiving the start bit respectively. The routine handles the next bit, before the routine exits. If the bit handled was the stop bit, the Timer/Counter overflow interrupt is disabled, and the external interrupt is again enabled.

9 “Extern0_interrupt( )” INTERRUPT SERVICE ROUTINE

The external interrupt 0 is active whenever the UART is idle. Upon an external interrupt, the “Extern0_interrupt( )” routine is called. This routine initiates the reception of serial data (an alternative function name would be: “uart_reception”). An external interrupt occurs on a falling edge on the 'INT0 pin (a falling edge marks the beginning of the start-bit – see Figure 1-1). This activates the Timer/Counter overflow interrupt and generates a 1.5 bit delay for the first start bit. Before exiting, the external interrupt is disabled to prevent falling edges in the incoming byte from reinitializing the receiver.

Figure 9-1. “Extern0_interrupt” Flow Chart.



10 Example Program

There is an example program included in this application note. The program will wait for a character. Upon reception, the software UART returns the message: 'atmel avr' if ‘a’ was sent. ‘Atmel avr’ if ‘A’ was sent and ‘Unknown command’ for all other cases.

11 Summary

In this application note, a software UART has been implemented. The MCU is capable of using 38400 baud at 8 MHz clock frequency. The UART is initialized by calling “initSoftwareUart()” and enabling global interrupts. If the UART is idle, it will automatically receive incoming data. To transmit data, a subroutine called “putChar()” is called with the data passed to the function as an unsigned char.

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

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© **2008 Atmel Corporation. All rights reserved**. Atmel®, logo and combinations thereof, and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

0941C-AVR-04/08