AVR304: Half Duplex Interrupt Driven Software UART

Features

- **Runs asynchronously (interrupt driven)**
- **Capable of handling baud rates of up to 38.4 kbaud @1 MHz XTAL**
- **Runs on any AVR device with 8-bit timer/counter and external interrupt**

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.

Theory of Operation

Asynchronous serial data communication follow some rules 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 a stop bits. This construction is called a frame. The frame format shown in Figure 1 and Figure 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.

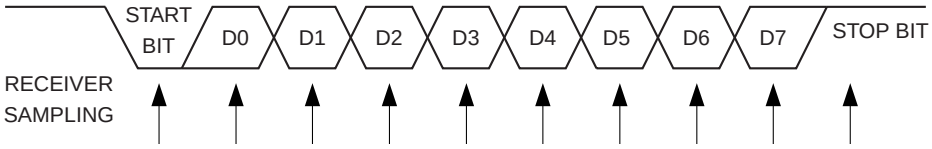


Figure 1. UART Communication Frame Format

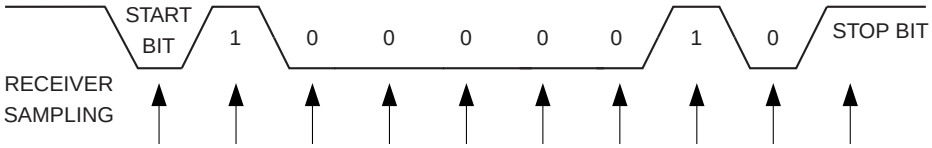


Figure 2. Serial frame of ASCII “A” ($41)

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.

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.

8-Bit AVR Microcontroller

Application Note

0941A-A–8/97

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 has to be set up using the same baud rate equally for correct reception.

As seen in Figure 1, the frame starts with a falling edge. This falling edge generates an initial interrupt (using external interrupt). The interrupt starts Timer/Counter0, and preset it to time out in exactly 1.5 bit lengths. This 1.5 bit length delay is required to generate the next sampling event at the bit-center of the first data bit. The next 8 interrupts are generated by a predefined delay (1 bit length) using counter/timer 0. Figure 3 shows the flowchart for receiving serial data.

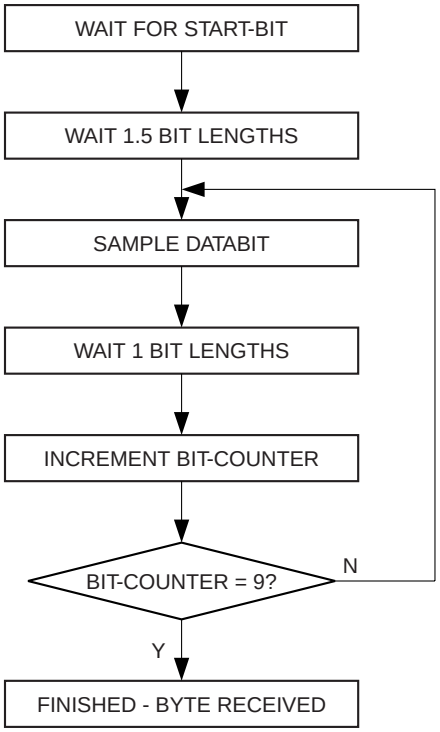


Figure 3. Flowchart for receiving serial data

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 up. 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 (if not, the receiver can't separate the data bytes). This is always a logical one (or mark). Figure 4 shows the flowchart for transmitting serial data.

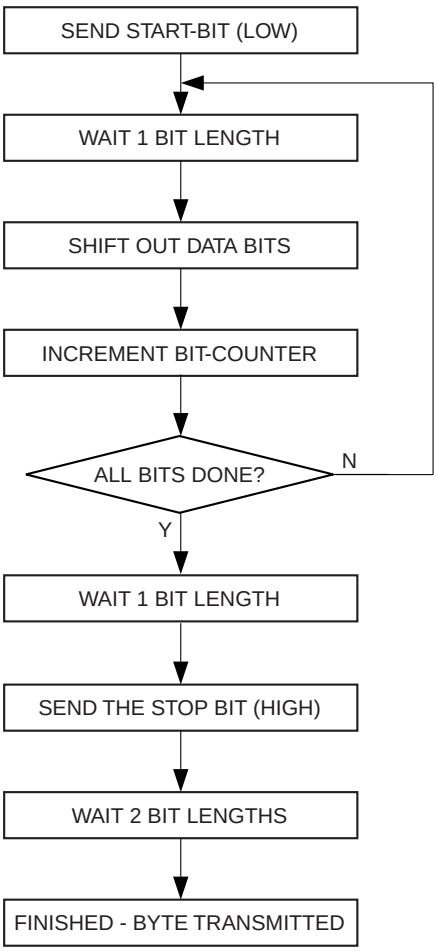
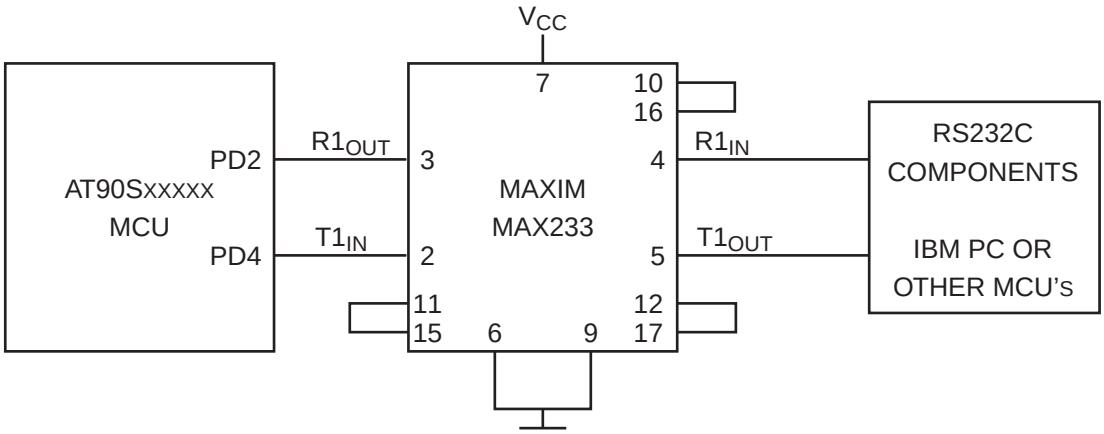


Figure 4. Flowchart for transmitting serial data

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 onboard DC-DC converter to convert the 5 volts to the RS-232 signaling levels.

The receive pin must be connected to PD2 because of the external interrupt. It is not important which pin is used as the transmit pin, and this application note will use PD4. Figure 5 shows how the MCU should physically be connected to an RS-232 line.



Note: See Maxim's Databook for further information

Figure 5. Physical connection to an RS232 serial line

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 1MHz 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. 256-N is the value pre stored in the timer/counter, and C is the Timer/Counter 0 prescaling factor, as described in the T/C Prescaler in the AVR databook,. 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. The absolute minimum value of N•C is 17. If N•C is set to be smaller, the overflow will occur even before the T/C interrupt handler has finished). Absolute maximum value of N is (170+20/C), as the receiver has to generate a delay of 1.5 bit periods when receiving the start bit.

$$\frac{17}{C} \le N \le 170 + \frac{20}{C}$$

"UART_INIT" SUBROUTINE - INITIALISE UART

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

Table 1. “uart_init” Subroutine Performance Figures

Parameter	Value
Code Size	8 words
Execution cycles	11, including the RET instruction
Register Usage	• Low registers :None • High registers :2 • Global registers :1 • Pointers :None

Table 2. “uart_init” Register Usage

Register	Input	Internal	Output
R16		“u_tmp” - Scratch register	
R18			“u_status” - Erased status-registers (used by the others routines) - read only

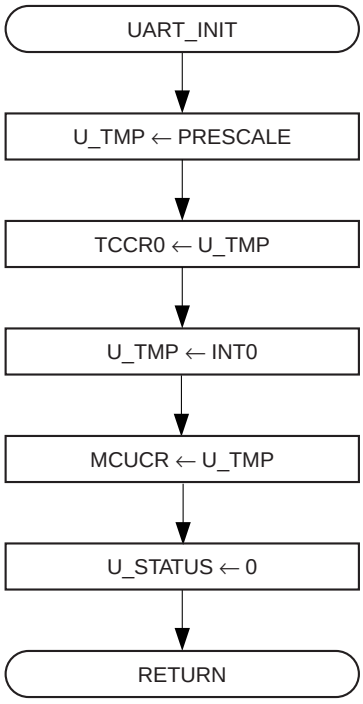


Figure 6. “uart_init” Flow Chart

“UART_TRANSMIT” SUBROUTINE - TRANSMITTING A BYTE

This routine is used when the software wish to transmit data. It sets the transmit- and the busy-flags, disables external interrupt (this disables reception when transmitting), sets the correct baud rate (t/c0 interrupt) and sets the start bit.

NB! This routine cancels all other pending activities. Calling “uart_transmit” will clear the UART shift register “u_buffer”, clearing any data currently stored in this register. If called while receiving data, the transmitted data may be corrupted if a Timer/Counter overflow interrupt occurs while executing the subroutine. By waiting until the BUSY flag is cleared in “u_status”, safe transmissions are guaranteed. The use of “u_status” is explained in detail below.

Table 3. "uart_transmit" Subroutine Performance Figures

Parameter	Value
Code Size	13 words
Execution Cycle	17 (including RET)
Register Usage	• Low registers :1 • High registers :4 • Global registers :3 • Pointers :None

Table 4. "uart_transmit" Register Usage

Register	Input	Internal	Output
R20	"u_transmit" - Byte to be transmitted. When external interrupt is disabled, contents are copied into "u_buffer".		
R19		"u_reload" - to save time in the interrupt routine, this register contains the timer reload value.	
R18			u_status" - This register is used to indicate the different states of the UART. See separate section for details. This subroutine sets the transmit- and busy-flags.
R14		"u_buffer" -UART shift register. When the UART is locked by this subroutine, the value from "u_transmit" is copied here.	
R17		"u_bit_cnt" - bit counter, reset by this subroutine.	
R16		"u_tmp" - A scratch register.	

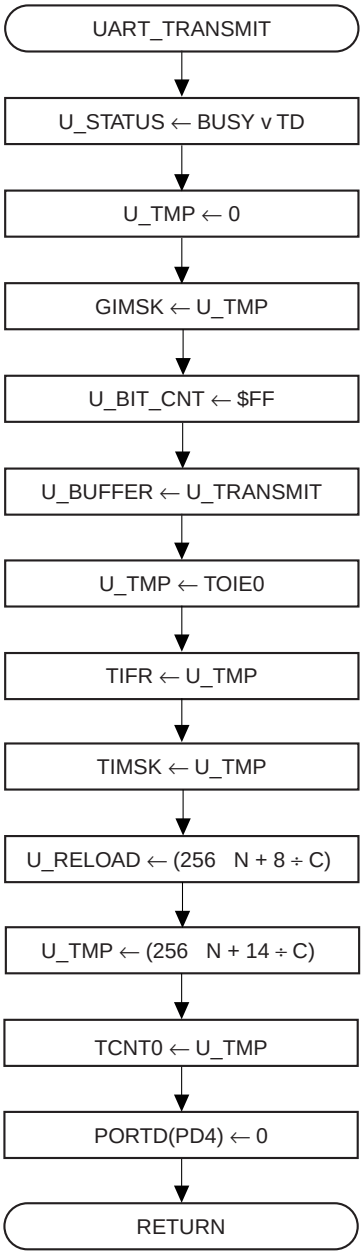


Figure 7. Flow Chart for “uart_transmit”

“U_STATUS” REGISTER BYTE

The u_status byte is used by all functions. It have three bits implemented:

1. “BUSY”. This bit indicates when the UART is busy. If only the busy bit is set, the UART is currently receiving data.
2. “TD”. Transmitting Data. This bit is set in conjunction with the Busy-flag. If this bit is set, the UART is currently transmitting data. The bit is set by “uart_transmit”, and automatically cleared after the stop bit has been sent in “tim0_ovf”.
3. “RDR”. Receive Data Ready. This bit is set whenever “u_buffer” contains valid received data. When the UART begins to transmit or receive, the bit will be cleared. The bit can also be cleared by software after received data has been read from “u_buffer”.

Software will detect new incoming data is present by reading a one in the RDR-bit. Whenever this bit goes high, new data has arrived. If the software is waiting for available time to send data, he can read the BUSY flag, and call “uart_transmit” whenever this flag is cleared.

The “u_status” byte is read-only and should not be altered by user software. To clear the RDR bit however, software is allowed to do so by using a “cbr” (Clear Bit in Register) instruction operating directly on the “u_status” register.

“TIM0_OVF” INTERRUPT SERVICE ROUTINE

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

The Timer/Counter overflow interrupt is enabled by “uart_transmit” or “ext0_int”, when transmitting or receiving the start bit respectively. Upon entering, the Timer/Counter is preset to give the next overflow in one bit length. Then the next bit is handled, before the routine exits. If the bit handled was the stop bit, the Timer/Counter overflow interrupt is cleared, and the external interrupt is again enabled.

Table 5. "tim0_ovf" Interrupt Service Routine Performance Figures

Parameter	Value
Code Size	35 words
Execution Cycles	min. 18, max. 28 - including the "reti" instruction. Varies according to task (read/write - data/stop/start-bit - etc.)
Register Usage	• Low registers :2 • High registers :4 • Global registers :4 • Pointers :None
Interrupt Usage	Timer/Counter 0 overflow interrupt

Table 6. "tim0_ovf" Register Usage

Register	Input	Internal	Output
R15		Internal SREG storage	
R19		"u_reload" - to save time in the interrupt time, this register contains the timer reload value.	
R18	"u_status" - uses this register to determine what task to perform		
R14		"u_buffer" - the UART shift-register for received and transmitted data.	
R17		"u_bit_cnt" - bit counter.	
R16		"u_tmp" - Scratch register	

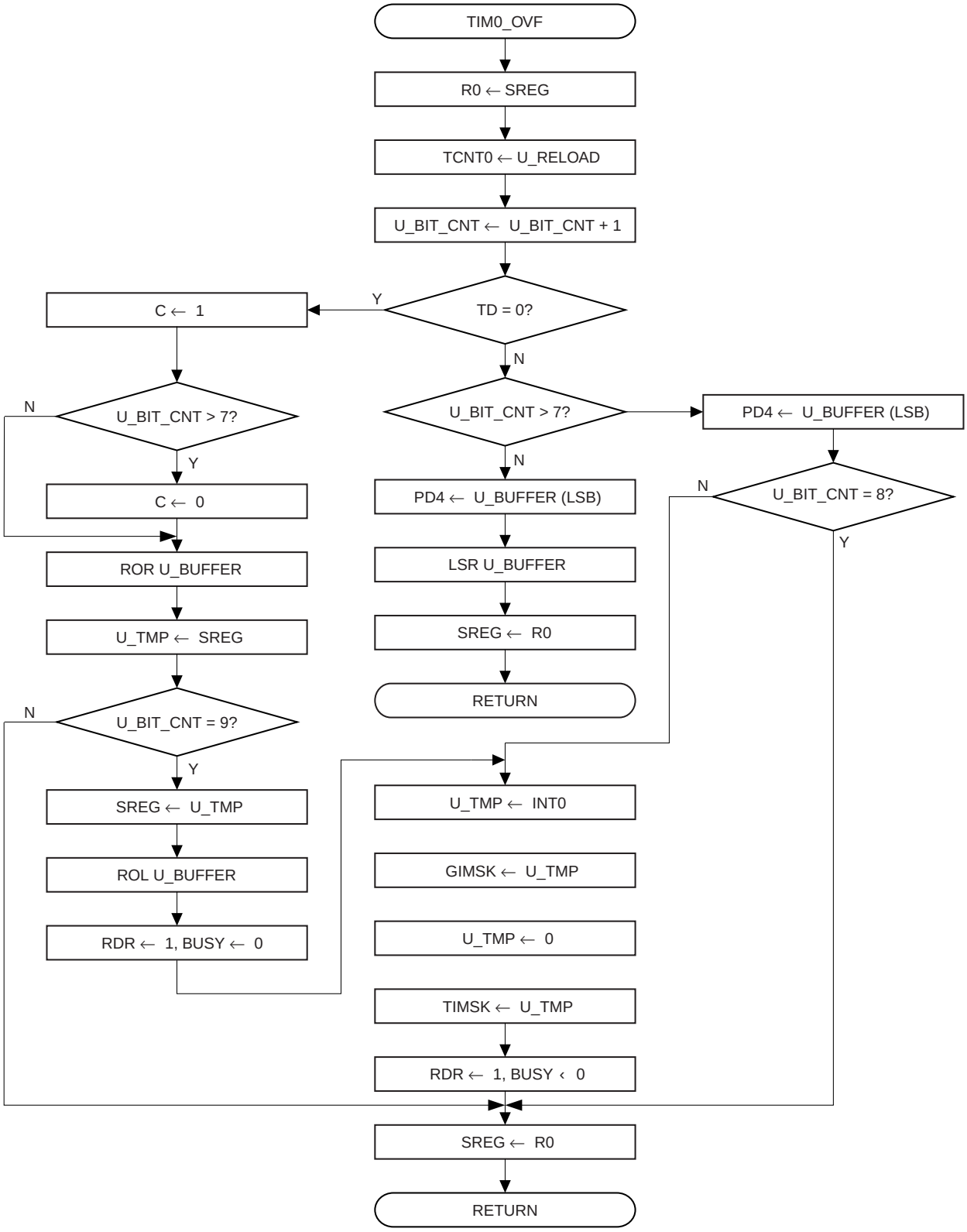


Figure 8. Flow Chart for “tim0_ovf”

“EXT_INT0” INTERRUPT SERVICE ROUTINE

The external interrupt 0 is active whenever the UART is idle. Upon an external interrupt, the “ext_int0” routine is called. This routine initiates the reception of serial data (an alternative 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 fig. 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.

Table 7. “ext_int0” Interrupt Service Routine Performance Figures

Parameter	Value
Code Size	12 words
Execution Cycles	15, including reti
Register Usage	• Low registers :1 • High registers :4 • Global registers :3 • Pointers :None
Interrupt Usage	External Interrupt INT0

Table 8. “ext_int0” Register Usage

Register	Input	Internal	Output
R15		Status Register Temporary Storage	
R19		“u_reload” - this register contains the timer reload value.	
R18			u_status” - This register is used to indicate the different states of the UART. See separate section for details. This subroutine sets (unconditionally) the busy-flag and clears the transmit and receive ready flags.
R17		“u_bit_cnt” - bit counter, reset by this routine	
R16		“u_tmp” -scratch register	

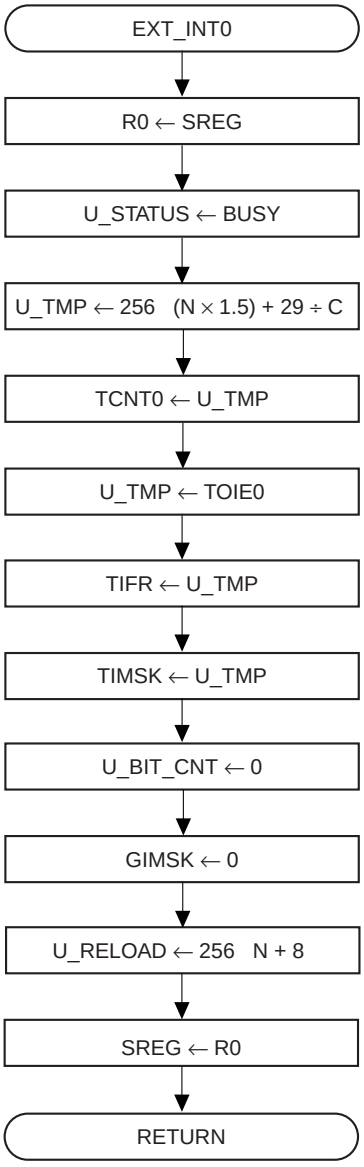


Figure 9. Flow chart for “ext_int0”

Tips and Warnings

In the implementation, some IO-registers are manipulated without preserving current register settings. In the case of GIMSK, GIFR, TIMSK, and TIFR, altering these registers might also affect the operation of other peripherals. Software should normally manipulate the bits needed, preserving the rest, but to speed up the UART, the routines sets these registers by brute force. Any other bits that were in use, will be cleared. If other peripherals are being used, all UART routines must be extended to preserve all other flags.

Example Program

There is an example program included in this application note. The program will wait for a character. Upon reception, the received data is presented on port B, Simultaneously, the software UART returns the message: 'You typed <character>'.

Performance Figures

Table 9. Overall Performance Figures

Parameter	Value
Code Size	72 words - UART routines only 100 words - complete application note
Register Usage	• Low Registers :2 • High Registers :5 • Pointers :None
Interrupt Usage	Timer/Counter 0 Interrupt External Interrupt 0
Peripheral Usage	Timer/Counter0 Port B, all pins (example program only) Port D, pin 2 and 4 EEPROM (example program only)

Summary

In this application note, a software UART has been implemented. The MCU is capable of using 38400 baud at 1 MHz crystal. The UART is initialized by calling “uart_init” and enabling global interrupts. If the UART is idle, it will automatically receive incoming data. To transmit data, a subroutine called “uart_transmit” is called with the data to send stored in the “u_transmit”.