

Simple Serial Communications With AVR libc

The light and fast alternative.

I like to use various Arduino boards for AVR development. What I do not like are the Arduino libraries. They are often just wrappers around libc functions or rewrites of functions libc already provides. Serial communications is one good example. Arduino provides you with its own implementation of `Serial.print()`, `Serial.println()` and `Serial.read()` methods. At the same time AVR Libc has proven `printf()`, `puts()` and `getchar()` functions. This article explains easy implementation of libc functions used for serial communications.

If you do not have much experience in programming it is probably better to stick with Arduino libraries. They are good at hiding some of the confusing features of embedded programming. However changes are you grow out of them after few projects. Atmel datasheets are not as confusing as they first appear. You might also want to check the finished code of this article.

Configuring UART

AVR microcontrollers have three control and status registers. Register `UCSR0A` mostly contains status data. `UCSR0B` and `UCSR0C` contain all the configuration settings. See the tables in the end of article for all possible values.

AVR Libc provides helper macros for baud rate calculations. Header file requires `F_CPU` and `BAUD` to be defined. After including the header file `UBRRL_VALUE`, `UBRRH_VALUE` and `USE_2X` are defined. First two are used to set UART speed. Last one is used to determine if UART has to be configured to run in double speed mode with given baud rate.

`UCSZ20`, `UCSZ01` and `UCSZ00` control the data size. Possible sizes are 5-bit (000), 6-bit (001), 7-bit (010), 8-bit (011) and 9-bit (111). Most common used data size is 8-bit.

With above bits we can set most common configuration: no parity, 8 data bits, 1 stop bit.

```

#define F_CPU 16000000UL
#define BAUD 9600

#include <util/setbaud.h>

void uart_init(void) {
    UBRRH = UBRRH_VALUE;
    UBRR0L = UBRR0L_VALUE;

    #if USE_2X
        UCSRA |= _BV(U2X0);
    #else
        UCSRA &= ~(_BV(U2X0));
    #endif

    UCSRC = _BV(UCSZ01) | _BV(UCSZ00); /* 8-bit data */
    UCSRB = _BV(RXEN0) | _BV(TXEN0); /* Enable RX and TX */
}

```

Writing and Reading From UART

You can transmit data to UART by writing a byte to USART Data Register `UDR0`. First you have to make sure UART is ready to transmit new data. You can wait until USART Data Register Empty `UDRE` flag is set. Alternatively you can wait after each byte to transmission be ready. USART Transmit Complete `TXC0` is set when transmission is ready.

```

void uart_putchar(char c) {
    loop_until_bit_is_set(UCSRA, UDRE0); /* Wait until data register empty. */
    UDR0 = c;
}

void uart_putchar(char c) {
    UDR0 = c;
    loop_until_bit_is_set(UCSRA, TXC0); /* Wait until transmission ready. */
}

```

You can receive data from UART by reading a byte from USART Data Register `UDR0`. USART Receive Complete `RXC0` flag is set if to unread data exists in data register.

```

char uart_getchar(void) {
    loop_until_bit_is_set(UCSRA, RXC0); /* Wait until data exists. */
    return UDR0;
}

```

Redirecting STDIN and STDOUT to UART

`FDEV_SETUP_STREAM` macro can be used to setup a buffer which is valid for stdio operations. Initialized buffer will be of type `FILE`. You can define separate buffers for input and output. Alternatively you can define only one buffer which works for both input and

output. First and second parameters are names of the functions which will be called when data is either read from or written to the buffer.

```
FILE uart_output = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);
FILE uart_input = FDEV_SETUP_STREAM(NULL, uart_getchar, _FDEV_SETUP_READ);

FILE uart_io FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);
```

To prepare our `uart_putchar` and `uart_getchar` function to be used with streams we have to change the definition a bit. To properly format output we also force adding a carriage return after newline has been sent.

```
void uart_putchar(char c, FILE *stream) {
    if (c == '\n') {
        uart_putchar('\r', stream);
    }
    loop_until_bit_is_set(UCSR0A, UDRE0);
    UDR0 = c;
}

char uart_getchar(FILE *stream) {
    loop_until_bit_is_set(UCSR0A, RXC0); /* Wait until data exists. */
    return UDR0;
}
```

Now we can redirect both STDIN and STDOUT to UART. This enables us to use AVR Libc provided functions to read and write to serial port.

```
int main(void) {

    uart_init();
    stdout = &uart_output;
    stdin = &uart_input;

    char input;

    while(1) {
        puts("Hello world!");
        input = getchar();
        printf("You wrote %c\n", input);
    }

    return 0;
}
```

Control and Status Registers

UCSR0A Bit #	Name	Description
bit 7	RXC0	USART Receive Complete. Set when data is available and the data register has not be read yet.
bit 6	TXC0	USART Transmit Complete. Set when all data has transmitted.
bit 5	UDRE0	USART Data Register Empty. Set when the UDR0 register is empty and new data can be transmitted.
bit 4	FE0	Frame Error. Set when next byte in the UDR0 register has a framing error.
bit 3	DOR0	Data OverRun. Set when the UDR0 was not read before the next frame arrived.
bit 2	UPE0	USART Parity Error. Set when next frame in the UDR0 has a parity error.
bit 1	U2X0	USART Double Transmission Speed. When set decreases the bit time by half doubling the speed.

UCSR0A Bit #	Name	Description
bit 0	MPCM0	Multi-processor Communication Mode. When set incoming data is ignored if no addressing information is provided.

UCSR0B Bit #	Name	Description
bit 7	RXCIE0	RX Complete Interrupt Enable. Set to allow receive complete interrupts.
bit 6	TXCIE0	TX Complete Interrupt Enable. Set to allow transmission complete interrupts.
bit 5	UDRIE0	USART Data Register Empty Interrupt Enable. Set to allow data register empty interrupts.
bit 4	RXEN0	Receiver Enable. Set to enable receiver.
bit 3	TXEN0	Transmitter enable. Set to enable transmitter.
bit 2	UCSZ20	USART Character Size 0. Used together with UCSZ01 and UCSZ00 to set data frame size. Available sizes are 5-bit (000), 6-bit (001), 7-bit (010), 8-bit (011) and 9-bit (111).

UCSR0B Bit #	Name	Description
bit 1	RXB80	Receive Data Bit 8. When using 8 bit transmission the 8th bit received.
bit 0	TXB80	Transmit Data Bit 8. When using 8 bit transmission the 8th bit to be submitted.

UCSR0C Bit #	Name	Description
bit 7 bit 6		USART Mode Select 1 and 0. UMSEL01 and UMSEL00 combined select the operating mode. Available modes are asynchronous (00), synchronous (01) and master SPI (11).
bit 5 bit 4	UPM01 UPM00	USART Parity Mode 1 and 0. UPM01 and UPM00 select the parity. Available modes are none (00), even (10) and odd (11).
bit 3	USBS0	USART Stop Bit Select. Set to select 1 stop bit. Unset to select 2 stop bits.
bit 2 bit 1	UCSZ01 UCSZ00	USART Character Size 1 and 0. Used together with UCSZ20 to set data frame size. Available sizes are 5-bit (000), 6-bit (001), 7-bit (010), 8-bit (011) and 9-bit (111).

UCSR0C Bit #	Name	Description
bit 0	UCPOL0	USART Clock Polarity. Set to transmit on falling edge and sample on rising edge. Unset to transmit on rising edge and sample on falling edge.

More Reading

[Full source code](#) of this article. [USART entry in QEEWiki](#) by Q. [Serial Communication Using AVR Microcontroller USART](#) by Engineers Garage. [Serial Communication Notes](#) by Rod Byrne.

Want to improve your understanding of electronics? Try the [Basic Electronics for Arduino Makers](#) video course from Udemy.

Posted in [AVR Electronics](#) on 19 Nov 2011.



Copyright © 2017 Mika Tuupola