Ringbuffer\_Code:

<https://www.mikrocontroller.net/articles/FIFO>

<https://en.wikipedia.org/wiki/Circular_buffer>

#include **"uart.h"**

#include **<avr/io.h>**

#include **<avr/pgmspace.h>**

#include **<avr/interrupt.h>**

#define SEND\_BUF\_BITS **4**

#define SEND\_BUF\_SIZE (**1**<<SEND\_BUF\_BITS)

#define SEND\_BUF\_MASK (SEND\_BUF\_SIZE-**1**)

#define RECV\_BUF\_BITS **2**

#define RECV\_BUF\_SIZE (**1**<<RECV\_BUF\_BITS)

#define RECV\_BUF\_MASK (RECV\_BUF\_SIZE-**1**)

**static** **volatile** uint8\_t send\_buf\_start;

**static** uint8\_t send\_buf\_end;

**static** **char** send\_buf[SEND\_BUF\_SIZE];

**static** uint8\_t recv\_buf\_start;

**static** **volatile** uint8\_t recv\_buf\_end;

**static** **char** recv\_buf[RECV\_BUF\_SIZE];

**void** uart\_init(uint16\_t baud) {

// ifdef ATTINY...

UCSRA = \_BV(U2X);

UCSRB = \_BV(TXEN) | \_BV(RXEN) | \_BV(RXCIE); //UART TX+RX einschalten

UCSRC |= \_BV(UCSZ0)|\_BV(UCSZ1); //Asynchron 8N1

UBRRH=(uint8\_t)((baud)>>**8**);

UBRRL=(uint8\_t)(baud);

send\_buf\_start=**0**;

send\_buf\_end=**0**;

recv\_buf\_start=**0**;

recv\_buf\_end=**0**;

}

**void** uart\_putc(**char** c)

{

uint8\_t next\_pos=(send\_buf\_end+**1**)&SEND\_BUF\_MASK;

// wait till we got space in buffer

**while** ((SREG & \_BV(SREG\_I)) && (next\_pos == send\_buf\_start));

**if** (next\_pos == send\_buf\_start) {

// Interrupts are disabled and buffer is full. we loose this char.

**return**;

}

send\_buf[send\_buf\_end]=c;

send\_buf\_end=next\_pos;

// Enable UDR interrupt. will immediatly call ISR if UDRE is set.

UCSRB = \_BV(TXEN) | \_BV(RXEN) | \_BV(UDRIE) | \_BV(RXCIE);

}

**void** uart\_write(**const** **char** \* c) {

**while** (\*c) {

uart\_putc(\*c++);

}

}

**void** uart\_write\_P(**const** **char** \* c) {

**char** ch;

**do** {

ch=pgm\_read\_byte(c++);

**if** (!ch) **return**;

uart\_putc(ch);

} **while**(**1**);

}

**char** uart\_getc(**void**) {

// Wait till byte in buffer

**while**(recv\_buf\_start == recv\_buf\_end);

**char** x=recv\_buf[recv\_buf\_start++];

recv\_buf\_start &=RECV\_BUF\_MASK;

**return** x;

}

uint8\_t uart\_recv\_buf\_size(**void**) {

**if** (recv\_buf\_end >= recv\_buf\_start) {

**return** recv\_buf\_end - recv\_buf\_start;

}

**return** recv\_buf\_end + RECV\_BUF\_SIZE - recv\_buf\_start;

}

uint8\_t uart\_send\_buf\_size(**void**) {

**if** (send\_buf\_end >= send\_buf\_start) {

**return** send\_buf\_end - send\_buf\_start;

}

**return** send\_buf\_end + SEND\_BUF\_SIZE - send\_buf\_start;

}

ISR(USART\_RX\_vect) {

recv\_buf[recv\_buf\_end]=UDR;

recv\_buf\_end=(recv\_buf\_end+**1**)&RECV\_BUF\_MASK;

**if** (recv\_buf\_end == recv\_buf\_start) {

// Overflow! drop the first byte of the recv\_buf.

recv\_buf\_start=(recv\_buf\_start+**1**)&RECV\_BUF\_MASK;

}

}

ISR(USART\_UDRE\_vect) {

**if** (send\_buf\_end == send\_buf\_start) {

// No data to send, disable ISR.

UCSRB = \_BV(TXEN) | \_BV(RXEN) | \_BV(RXCIE);

} **else** {

UDR=send\_buf[send\_buf\_start++];

send\_buf\_start &=SEND\_BUF\_MASK;

}

}