Muhammad Sharjeel and Naafiul Hossain

115185427

115107623

Pre Lab 8: Circular Buffers and AVR128DB48 USART Modules

ESE 381 Section L02

Bench 7

Breadboard: K2

```c
/*
 * usart3_init_test.c
 *
 * Created: 4/9/2025 12:47:07 AM
 * Author : Naafiul Hossain
 */

#define F_CPU 4000000 // Frequency of the CPU in Hz
#define USART3_BAUD_RATE(BAUD_RATE) ((float)((F_CPU * 64 )/ (16 * (float) BAUD_RATE)) + .5) // Macro to calculate baud rate for USART

#include <avr/io.h> // Includes the definitions of the register names
#include <avr/interrupt.h> // Includes AVR interrupt definitions
#include <util/delay.h> // Includes functions for delays
#include <string.h> // Includes functions for string manipulation

volatile uint8_t cntrlcBM; // Variable to hold the USART control configuration mask

// Function to wait until the USART data register is empty (ready to transmit)
void waitTxReady(void)
{
    // Loop until the data register empty flag is set
    while (!(USART3.STATUS & USART_DREIF_bm))
    {
        asm volatile ("nop"); // Do nothing operation
    }
}

// Function to send a character over USART
void USART3_Send(char c)
{
    waitTxReady(); // Wait for the transmit buffer to be ready
    USART3.TXDATAL = c; // Put the character into the USART data register
}

// Function to initialize USART3
void USART3_Init(uint16_t baud, uint8_t data_bits, unsigned char parity )
{
    PORTB.DIR = 0x01; // Set the first pin of PORTB as output, others as input

    USART3.BAUD = (uint16_t)USART3_BAUD_RATE(baud); // Set the baud rate using the macro

    USART3.CTRLB = 0b11000000; // Enable receiver and transmitter

    cntrlcBM = 0x00; // Start with a default control mask of 0

    // Set the number of data bits
```

```
47      switch(data_bits) {
48          case 5:
49          cntrlcBM |= 0x00; // lsb data bits
50          break;
51          case 6:
52          cntrlcBM |= 0x01; // 6 data bits
53          break;
54          case 7:
55          cntrlcBM |= 0x02; // 7 data bits
56          break;
57          case 8:
58          cntrlcBM |= 0x03; // msb data bits
59          break;
60          default:
61          cntrlcBM |= 0x00; // Default to 5 data bits if invalid selection
62          break;
63      }
64
65      // Set parity mode
66      switch(parity) {
67          case 'D':
68          cntrlcBM |= 0x00; // Disabled parity
69          break;
70          case 'E':
71          cntrlcBM |= 0x20; // Even parity
72          break;
73          case 'O':
74          cntrlcBM |= 0x30; // Odd parity
75          break;
76          default:
77          cntrlcBM |= 0x00; // Default to no parity if invalid selection
78          break;
79      }
80
81      USART3.CTRLC = cntrlcBM; // Set the control mask to the USART control
          register
82  }
83
84  int main(void)
85  {
86      volatile char data = 'S'; // Data to send
87      USART3_Init(9600, 8, 'E'); // Initialize USART with 9600 baud, 8 data
          bits, even parity
88
89      while (1)
90      {
91          USART3_Send(data); // Send the data
92          _delay_ms(50); // Wait for 50 milliseconds
93      }
```

```
94  }
95
```

```c
/*
 * usart3_avr128_cir_buff.c
 * Added our Usart Int function from Task 1
 * Created: 4/9/2025 7:38:57 PM
 * Author : Naafiul Hossain
 */

/*
*
* circular_buffer_avr128_usart3.c
*
* Created: 3/1/2024 6:26:29 PM
* Author : kshort
*/


#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#define F_CPU 4000000   // CPU clock in Hz
#define USART3_BAUD_RATE(BAUD_RATE) ((float)(4000000 * 64 / (16 * (float)BAUD_RATE)) + 0.5)

/* UART Buffer Defines */
#define USART_RX_BUFFER_SIZE 16     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART_TX_BUFFER_SIZE 16     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART_RX_BUFFER_MASK ( USART_RX_BUFFER_SIZE - 1 )
#define USART_TX_BUFFER_MASK ( USART_TX_BUFFER_SIZE - 1 )

#if ( USART_RX_BUFFER_SIZE & USART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif
#if ( USART_TX_BUFFER_SIZE & USART_TX_BUFFER_MASK )
#error TX buffer size is not a power of 2
#endif

//#define RX_BUFFER_MARGIN ((uint8_t)(0.1 * USART_RX_BUFFER_SIZE + 0.5))

/* Static Variables */
static unsigned char USART_RxBuf[USART_RX_BUFFER_SIZE];
static  uint8_t USART_RxHead;  //orig. declared volatile - kls
static  uint8_t USART_RxTail;  //orig. declared volatile - kls
static unsigned char USART_TxBuf[USART_TX_BUFFER_SIZE];
static  uint8_t USART_TxHead;  //orig. declared volatile - kls
static  uint8_t USART_TxTail;  //orig. declared volatile - kls


volatile uint8_t cntrlcBM ;

```

```
49  //uint8_t counter = 0;
50
51  /* Function Prototypes */
52  void USART3_Init(uint16_t baud, uint8_t data_bits, unsigned char parity);
53  uint8_t USART3_Receive( void );
54  void USART3_Transmit( uint8_t data );
55  uint8_t DataInReceiveBuffer(void);
56
57
58  /* Main - a simple test program*/
59  int main( void )
60  {
61      USART_RxTail = 0x00;    //clear buffer indexes, not really necessary
62      USART_RxHead = 0x00;    //because they are automatically cleared since
63      USART_TxTail = 0x00;    //declared as global uninitialized variables
64      USART_TxHead = 0x00;
65
66      // SW0 pin an input, must be pressed to transfer data from Rx to Tx buffer
67      PORTB.DIR &= ~PIN2_bm;      // SW0 pin pushbutton input
68      PORTB.PIN2CTRL = 0x08;      //enable pull up
69      //USART3_Init();   // Initialize USART3
70      USART3_Init( 9600,8,'D');   // Initialize USART3
71      USART3.CTRLA |= USART_RXCIE_bm; /* Receive Complete Interrupt must be
          enabled */
72      sei();           // Enable global interrupts => enable USART interrupts
73      for( ; ; )        // Forever
74      {
75          //Uncomment next statement to have operation independent of SW0
76          USART3_Transmit( USART3_Receive() );
77
78          //Uncomment next statement have operation dependent on SW0
79          if (!(VPORTB_IN & PIN2_bm)) USART3_Transmit( USART3_Receive() );
80      }
81      return 0;
82  }
83
84
85
86  // Function to initialize USART3
87  void USART3_Init(uint16_t baud, uint8_t data_bits, unsigned char parity){
88
89      PORTB.DIR = 0x01; // make the whole port an input.; // make the single pin
          an output.
90
91      USART3.BAUD =  (uint16_t)USART3_BAUD_RATE(baud);      //baud rate
92
93      USART3.CTRLB =  0b11000000;    //transmitter and receiver enabling  as
          output
94
```

```
 95
 96      cntrlcBM =  0x00 ;    //frame format
 97      //data bits format:
 98
 99      switch(data_bits) {
100          case 5:
101          cntrlcBM |= 0x00;
102          break;
103          case 6:
104          cntrlcBM |= 0x01;
105          break;
106          case 7:
107          cntrlcBM |= 0x02;
108          break;
109          case 8:
110          cntrlcBM |= 0x03;
111          break;
112          default:
113          cntrlcBM |= 0x00; //not valid choice
114          break;
115      }
116
117      //stop bit mode:
118
119      cntrlcBM |= 0x00; //1 stop bit
120      //cntrlcBM |= 0x04; // 2 stop bits
121
122      //parity format:
123
124
125      switch(parity) {
126          case 'D':
127          cntrlcBM |= 0x00;
128          break;
129          case 'E':
130          cntrlcBM |= 0x20;
131          break;
132          case 'O':
133          cntrlcBM |= 0x30;
134          break;
135          default:
136          cntrlcBM |= 0x00; //not valid choice
137          break;
138      }
139
140  }
141
142
143  /* Interrupt handlers */
```

```
144
145 ISR (USART3_RXC_vect)       //Receive complete interrupt
146 {
147     uint8_t data;
148
149     //The following variable is not necessary if you are not going to take any
        action
150     //for an overflow that requires keeping the old index. Instead just use
151     //USART_RxHead instead of tmphead.
152     uint8_t tmphead;
153
154     cli();      // Clear global interrupt flag
155
156     /* Read the received data */
157     data = USART3.RXDATAL;
158
159     /* Calculate buffer index, increment and possibly roll over index */
160     tmphead = ( USART_RxHead + 1 ) & USART_RX_BUFFER_MASK;
161
162
163     //*********************************************************************
164     /*
165     The following condition could be changed to
166     if ( (tmphead >= (USART_RxTail + RX_BUFFER_MARGIN)) || (USART_RxTail >=
          (tmphead + RX_BUFFER_MARGIN)));
167     {
168     //Use flow control to stop flow of characters:
169     (a) hardware unasserts CTS
170     (b) software send XOFF
171     }
172     */
173     //*********************************************************************
174
175     if ( tmphead == USART_RxTail )
176     {
177         // ERROR! Receive buffer overflow
178     }
179
180     USART_RxBuf[tmphead] = data; // Store received data in buffer
181     //Alternate position B for USART_RxHead = tmphead;
182     USART_RxHead = tmphead;      // Store new index (was prev. in position A)
183     sei();      // re enable global interrupts
184 }
185
186
187 ISR (USART3_DRE_vect)
188 {
189     uint8_t tmptail;
190     cli();      // Clear global interrupts
```

```
191
192     /* Check if all data is transmitted */
193     if ( USART_TxHead != USART_TxTail )
194     {
195         // Calculate buffer index
196         tmptail = ( USART_TxTail + 1 ) & USART_TX_BUFFER_MASK;
197         USART_TxTail = tmptail;      // Store new index
198
199         USART3.TXDATAL = USART_TxBuf[tmptail];  // Start transmission
200     }
201     else
202     {
203         USART3.CTRLA &= ~(USART_DREIE_bm);     // Disable UDRE interrupt
204     }
205     sei();
206 }
207
208
209 /* Read function */
210 unsigned char USART3_Receive( void )
211 {
212     uint8_t tmptail;
213
214     while ( USART_RxHead == USART_RxTail );  /* Wait for incomming data */
215     tmptail = ( USART_RxTail + 1 ) & USART_RX_BUFFER_MASK;/* Calculate buffer
        index */
216     USART_RxTail = tmptail;                /* Store new index */
217     return USART_RxBuf[tmptail];           /* Return data */
218 }
219
220 /* Write function */
221 void USART3_Transmit( uint8_t data )
222 {
223     uint8_t tmphead;
224     /* Calculate buffer index */
225     tmphead = ( USART_TxHead + 1 ) & USART_TX_BUFFER_MASK; /* Wait for free
        space in buffer */
226     while ( tmphead == USART_TxTail );
227     USART_TxBuf[tmphead] = data;           /* Store data in buffer */
228     USART_TxHead = tmphead;                /* Store new index */
229
230     USART3.CTRLA |= USART_DREIE_bm;                    /* Enable UDRE
        interrupt */
231 }
232
233 uint8_t DataInReceiveBuffer( void )
234 {
235     return ( USART_RxHead != USART_RxTail ); /* Return 0 (FALSE) if the
        receive buffer is empty */
```

```
236  }
```

```c
/*
 * circular_buffer_avr128_usart3_Task3.c
 * This is the same as Task 2 but we commented out the conditional on SWO
 * Created: 4/10/2025 4:30:00 PM
 * Author : Naafiul Hossain
 */

/*
*
* circular_buffer_avr128_usart3.c
*
* Created: 3/1/2024 6:26:29 PM
* Author : kshort
*/


#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#define F_CPU 4000000   // CPU clock in Hz
#define USART3_BAUD_RATE(BAUD_RATE) ((float)(4000000 * 64 / (16 * (float)
  BAUD_RATE)) + 0.5)

/* UART Buffer Defines */
#define USART_RX_BUFFER_SIZE 16     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART_TX_BUFFER_SIZE 16     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART_RX_BUFFER_MASK ( USART_RX_BUFFER_SIZE - 1 )
#define USART_TX_BUFFER_MASK ( USART_TX_BUFFER_SIZE - 1 )

#if ( USART_RX_BUFFER_SIZE & USART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif
#if ( USART_TX_BUFFER_SIZE & USART_TX_BUFFER_MASK )
#error TX buffer size is not a power of 2
#endif

//#define RX_BUFFER_MARGIN ((uint8_t)(0.1 * USART_RX_BUFFER_SIZE + 0.5))

/* Static Variables */
static unsigned char USART_RxBuf[USART_RX_BUFFER_SIZE];
static  uint8_t USART_RxHead;  //orig. declared volatile - kls
static  uint8_t USART_RxTail;  //orig. declared volatile - kls
static unsigned char USART_TxBuf[USART_TX_BUFFER_SIZE];
static  uint8_t USART_TxHead;  //orig. declared volatile - kls
static  uint8_t USART_TxTail;  //orig. declared volatile - kls


volatile uint8_t cntrlcBM ;

```

```
49  //uint8_t counter = 0;
50
51  /* Function Prototypes */
52  void USART3_Init(uint16_t baud, uint8_t data_bits, unsigned char parity);
53  uint8_t USART3_Receive( void );
54  void USART3_Transmit( uint8_t data );
55  uint8_t DataInReceiveBuffer(void);
56
57
58  /* Main - a simple test program*/
59  int main( void )
60  {
61      USART_RxTail = 0x00;    //clear buffer indexes, not really necessary
62      USART_RxHead = 0x00;    //because they are automatically cleared since
63      USART_TxTail = 0x00;    //declared as global uninitialized variables
64      USART_TxHead = 0x00;
65
66      // SW0 pin an input, must be pressed to transfer data from Rx to Tx buffer
67      PORTB.DIR &= ~PIN2_bm;      // SW0 pin pushbutton input
68      PORTB.PIN2CTRL = 0x08;      //enable pull up
69      //USART3_Init();   // Initialize USART3
70      USART3_Init( 9600,8,'D');   // Initialize USART3
71      USART3.CTRLA |= USART_RXCIE_bm; /* Receive Complete Interrupt must be
          enabled */
72      sei();           // Enable global interrupts => enable USART interrupts
73      for( ; ; )        // Forever
74      {
75          //Uncomment next statement to have operation independent of SW0
76          USART3_Transmit( USART3_Receive() );
77
78          //Uncomment next statement have operation dependent on SW0
79          //if (!(VPORTB_IN & PIN2_bm)) USART3_Transmit( USART3_Receive() );
80      }
81      return 0;
82  }
83
84
85
86  // Function to initialize USART3
87  void USART3_Init(uint16_t baud, uint8_t data_bits, unsigned char parity){
88
89      PORTB.DIR = 0x01; // make the whole port an input.; // make the single pin
          an output.
90
91      USART3.BAUD =  (uint16_t)USART3_BAUD_RATE(baud);      //baud rate
92
93      USART3.CTRLB =  0b11000000;    //transmitter and receiver enabling  as
          output
94
```

```
 95
 96      cntrlcBM =  0x00 ;    //frame format
 97      //data bits format:
 98
 99      switch(data_bits) {
100          case 5:
101          cntrlcBM |= 0x00;
102          break;
103          case 6:
104          cntrlcBM |= 0x01;
105          break;
106          case 7:
107          cntrlcBM |= 0x02;
108          break;
109          case 8:
110          cntrlcBM |= 0x03;
111          break;
112          default:
113          cntrlcBM |= 0x00; //not valid choice
114          break;
115      }
116
117      //stop bit mode:
118
119      cntrlcBM |= 0x00; //1 stop bit
120      //cntrlcBM |= 0x04; // 2 stop bits
121
122      //parity format:
123
124
125      switch(parity) {
126          case 'D':
127          cntrlcBM |= 0x00;
128          break;
129          case 'E':
130          cntrlcBM |= 0x20;
131          break;
132          case 'O':
133          cntrlcBM |= 0x30;
134          break;
135          default:
136          cntrlcBM |= 0x00; //not valid choice
137          break;
138      }
139
140  }
141
142
143  /* Interrupt handlers */
```

```
144
145 ISR (USART3_RXC_vect)       //Receive complete interrupt
146 {
147     uint8_t data;
148
149     //The following variable is not necessary if you are not going to take any
        action
150     //for an overflow that requires keeping the old index. Instead just use
151     //USART_RxHead instead of tmphead.
152     uint8_t tmphead;
153
154     cli();      // Clear global interrupt flag
155
156     /* Read the received data */
157     data = USART3.RXDATAL;
158
159     /* Calculate buffer index, increment and possibly roll over index */
160     tmphead = ( USART_RxHead + 1 ) & USART_RX_BUFFER_MASK;
161
162
163     //************************************************************************
164     /*
165     The following condition could be changed to
166     if ( (tmphead >= (USART_RxTail + RX_BUFFER_MARGIN)) || (USART_RxTail >=
        (tmphead + RX_BUFFER_MARGIN)));
167     {
168     //Use flow control to stop flow of characters:
169     (a) hardware unasserts CTS
170     (b) software send XOFF
171     }
172     */
173     //************************************************************************
174
175     if ( tmphead == USART_RxTail )
176     {
177         // ERROR! Receive buffer overflow
178     }
179
180     USART_RxBuf[tmphead] = data; // Store received data in buffer
181     //Alternate position B for USART_RxHead = tmphead;
182     USART_RxHead = tmphead;      // Store new index (was prev. in position A)
183     sei();      // re enable global interrupts
184 }
185
186
187 ISR (USART3_DRE_vect)
188 {
189     uint8_t tmptail;
190     cli();      // Clear global interrupts
```

```
191
192     /* Check if all data is transmitted */
193     if ( USART_TxHead != USART_TxTail )
194     {
195         // Calculate buffer index
196         tmptail = ( USART_TxTail + 1 ) & USART_TX_BUFFER_MASK;
197         USART_TxTail = tmptail;      // Store new index
198
199         USART3.TXDATAL = USART_TxBuf[tmptail];  // Start transmission
200     }
201     else
202     {
203         USART3.CTRLA &= ~(USART_DREIE_bm);     // Disable UDRE interrupt
204     }
205     sei();
206 }
207
208
209 /* Read function */
210 unsigned char USART3_Receive( void )
211 {
212     uint8_t tmptail;
213
214     while ( USART_RxHead == USART_RxTail );  /* Wait for incomming data */
215     tmptail = ( USART_RxTail + 1 ) & USART_RX_BUFFER_MASK;/* Calculate buffer
        index */
216     USART_RxTail = tmptail;                /* Store new index */
217     return USART_RxBuf[tmptail];           /* Return data */
218 }
219
220 /* Write function */
221 void USART3_Transmit( uint8_t data )
222 {
223     uint8_t tmphead;
224     /* Calculate buffer index */
225     tmphead = ( USART_TxHead + 1 ) & USART_TX_BUFFER_MASK; /* Wait for free
        space in buffer */
226     while ( tmphead == USART_TxTail );
227     USART_TxBuf[tmphead] = data;           /* Store data in buffer */
228     USART_TxHead = tmphead;                /* Store new index */
229
230     USART3.CTRLA |= USART_DREIE_bm;                    /* Enable UDRE
        interrupt */
231 }
232
233 uint8_t DataInReceiveBuffer( void )
234 {
235     return ( USART_RxHead != USART_RxTail ); /* Return 0 (FALSE) if the
        receive buffer is empty */
```

```
236 }
```