EE-222: Microprocessor Systems

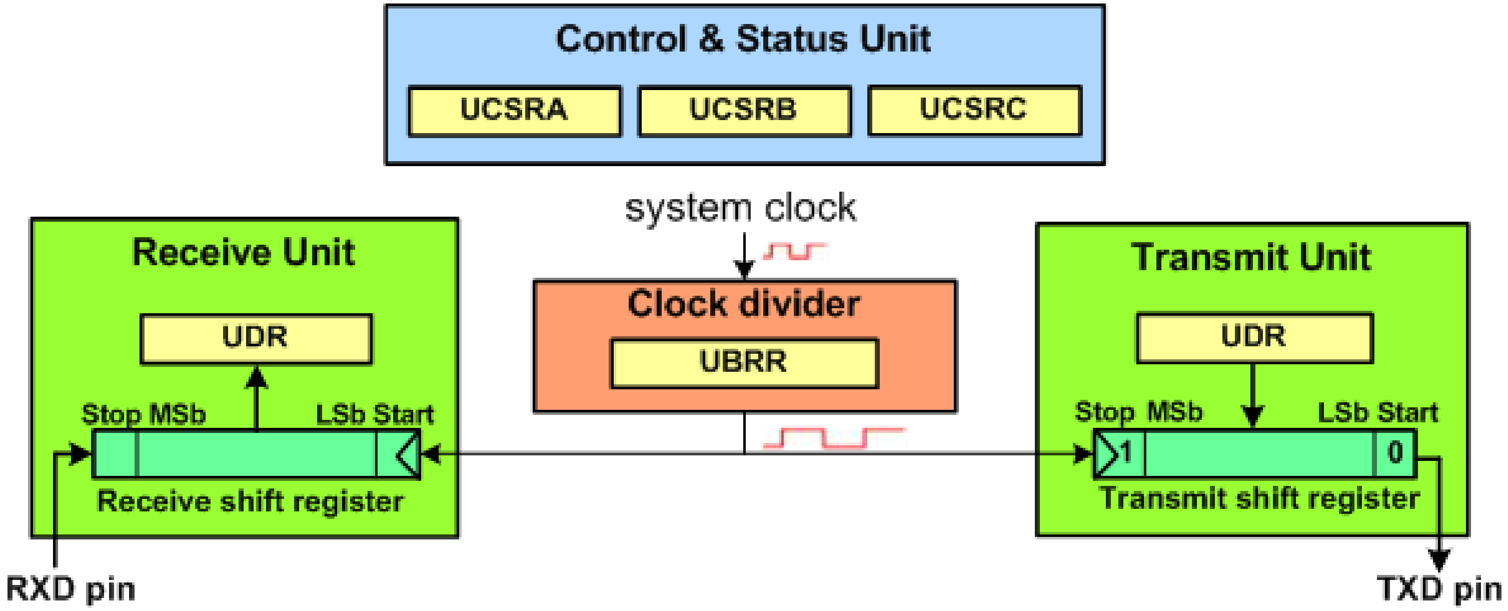
Programming AVR USART

Instructor: Dr. Arbab Latif

NUST
SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Last Lecture: UART in AVR

- Control registers: (initialize speed, data size, parity, etc)
  - UBRR, UCSRA, UCSRB, and UCSRC
- Send/receive register
  - UDR
- Status register
  - UCSRA



UCSRA

UCSRA:	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0

- RXC0 (Bit 7): USART Receive Complete 0
  - This flag bit is set when there are new data in the receive buffer that are not read yet. It is cleared when the receive buffer is empty. It also can be used to generate a receive complete interrupt.
- TXC0 (Bit 6): USART Transmit Complete 0
  - This flag bit is set when the entire frame in the transmit shift register has been transmitted and there are no new data available in the transmit data buffer register (TXB). It can be cleared by writing a one to its bit location. Also it is automatically cleared when a transmit complete interrupt is executed. It can be used to generate a transmit complete interrupt.
- UDRE0 (Bit 5): USART Data Register Empty 0
  - This flag is set when the transmit data buffer is empty and it is ready to receive new data. If this bit is cleared you should not write to UDR0 because it overrides your last data. The UDRE0 flag can generate a data register empty interrupt.
- FE0 (Bit 4): Frame Error 0
  - This bit is set if a frame error has occurred in receiving the next character in the receive buffer. A frame error is detected when the first stop bit of the next character in the receive buffer is zero.
- DOR0 (Bit 3): Data OverRun 0
  - This bit is set if a data overrun is detected. A data overrun occurs when the receive data buffer and receive shift register are full, and a new start bit is detected.
- PE0 (Bit 2): Parity Error 0
  - This bit is set if parity checking was enabled (UPM1 = 1) and the next character in the receive buffer had a parity error when received.
- U2X0 (Bit 1): Double the USART Transmission Speed 0
- MPCM0 (Bit 0): Multi-processor Communication Mode 0

UCSRB

UCSRB:	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80

- RXCIE0 (Bit 7): Receive Complete Interrupt Enable
  - To enable the interrupt on the RXC0 flag in UCSR0A you should set this bit to one.
- TXCIE0 (Bit 6): Transmit Complete Interrupt Enable
  - To enable the interrupt on the TXC0 flag in UCSR0A you should set this bit to one.
- UDRIE0 (Bit 5): USART Data Register Empty Interrupt Enable
  - To enable the interrupt on the UDRE0 flag in UCSR0A you should set this bit to one.
- RXEN0 (Bit 4): Receive Enable
  - To enable the USART receiver you should set this bit to one.
- TXEN0 (Bit 3): Transmit Enable
  - To enable the USART transmitter you should set this bit to one.
- UCSZ02 (Bit 2): Character Size
  - This bit combined with the UCSZ1:0 bits in UCSRC sets the number of data bits (character size) in a frame.
- RXB80 (Bit 1): Receive data bit 8
  - This is the ninth data bit of the received character when using serial frames with nine data bits.
- TXB80 (Bit 0): Transmit data bit 8
  - This is the ninth data bit of the transmitted character when using serial frames with nine data bits.

UCSRC

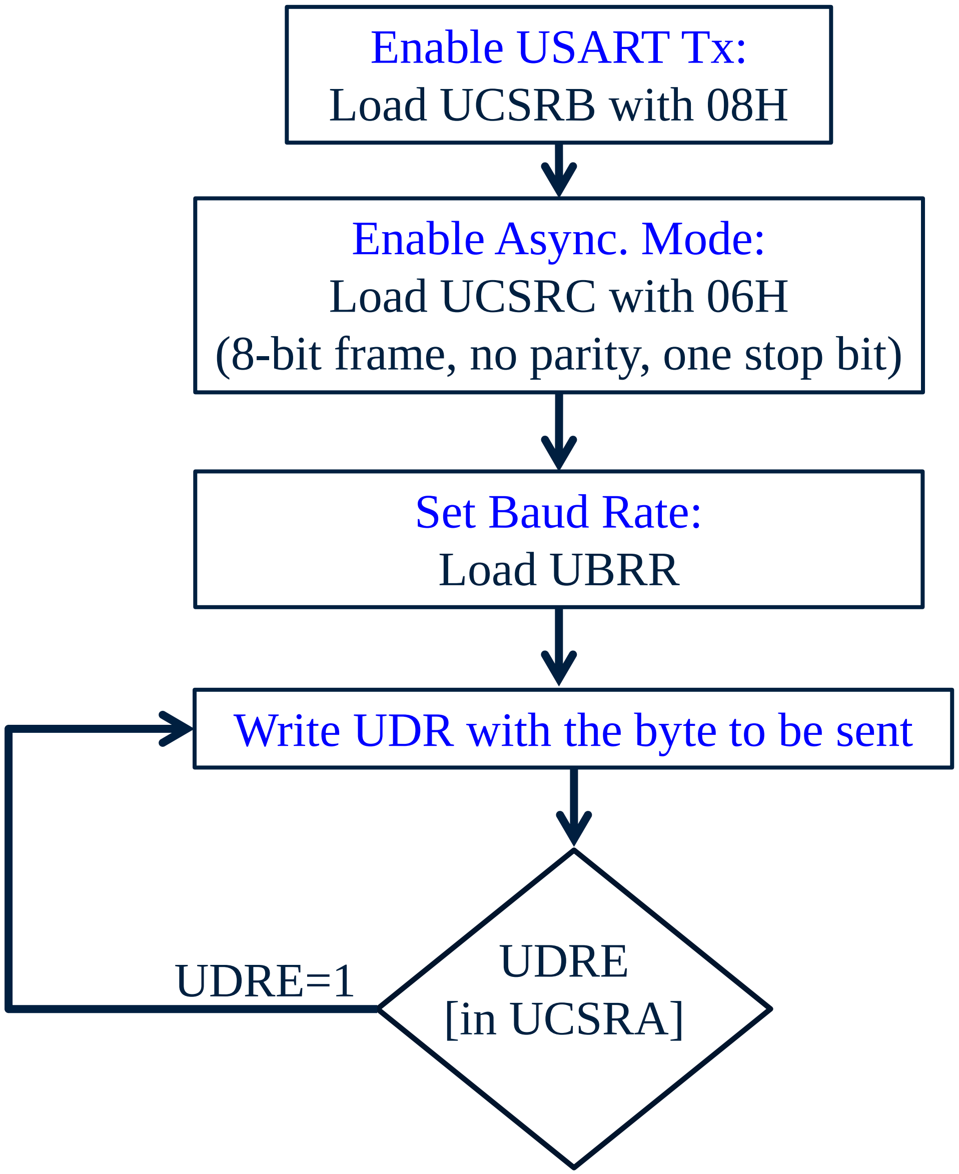
UCSRC:	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0

- UMSEL01:00 (Bits 7:6): USART Mode Select
  - These bits select the operation mode of the USART:
    - 00 = Asynchronous USART operation
    - 01 = Synchronous USART operation
    - 10 = Reserved
    - 11 = Master SPI (MSPIM)
- UPM01:00 (Bit 5:4): Parity Mode
  - These bits disable or enable and set the type of parity generation and check.
    - 00 = Disabled
    - 01 = Reserved
    - 10 = Even Parity
    - 11 = Odd Parity
- USBS0 (Bit 3): Stop Bit Select
  - This bit selects the number of stop bits to be transmitted.
    - 0 = 1 bit
    - 1 = 2 bits
- UCSZ01:00 (Bit 2:1): Character Size
  - These bits combined with the UCSZ02 bit in UCSR0B set the character size in a frame.
- UCPOL0 (Bit 0): Clock Polarity
  - This bit is used for synchronous mode.

UCSZ02	UCSZ01	UCSZ00	Char. size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	1	1	9-bit

Programming the AVR to Transfer Data Serially

Programming the AVR to Transfer Data Serially



Example: Transfer Data Serially

- Write a program for the AVR to transfer the letter ‘G’ serially at 9600 baud, continuously. Assume XTAL=8MHz.

```
   ldi r16, (1<<TXEN)                    ;enable transmitter
   out UCSRB, r16
   ldi r16, (1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL)    ;8-bit data
   out UCSRC, r16                    ;no parity, 1 stop bit
   ldi r16, 0x33                     ;9600 baud rate
   out UBRRL, r16                    ;for XTAL=8MHz
AGAIN:
   sbis UCSRA, UDRE                  ;is UDR empty
   rjmp AGAIN                        ;wait more
   ldi r16, 'G'                      ;send 'G'
   out UDR, r16                      ;to UDR
   rjmp AGAIN                        ;do it again
```

Can we do better in the previous example?

Interrupt-based Data Transmit

- Set UDRIE in UCSRB:
  - Setting this bit enables the interrupt on the UDRE flag in UCSRA
    - When the UDR register is ready to accept new data, the UDRE flag is set

Example: Interrupt-based Data Transmit

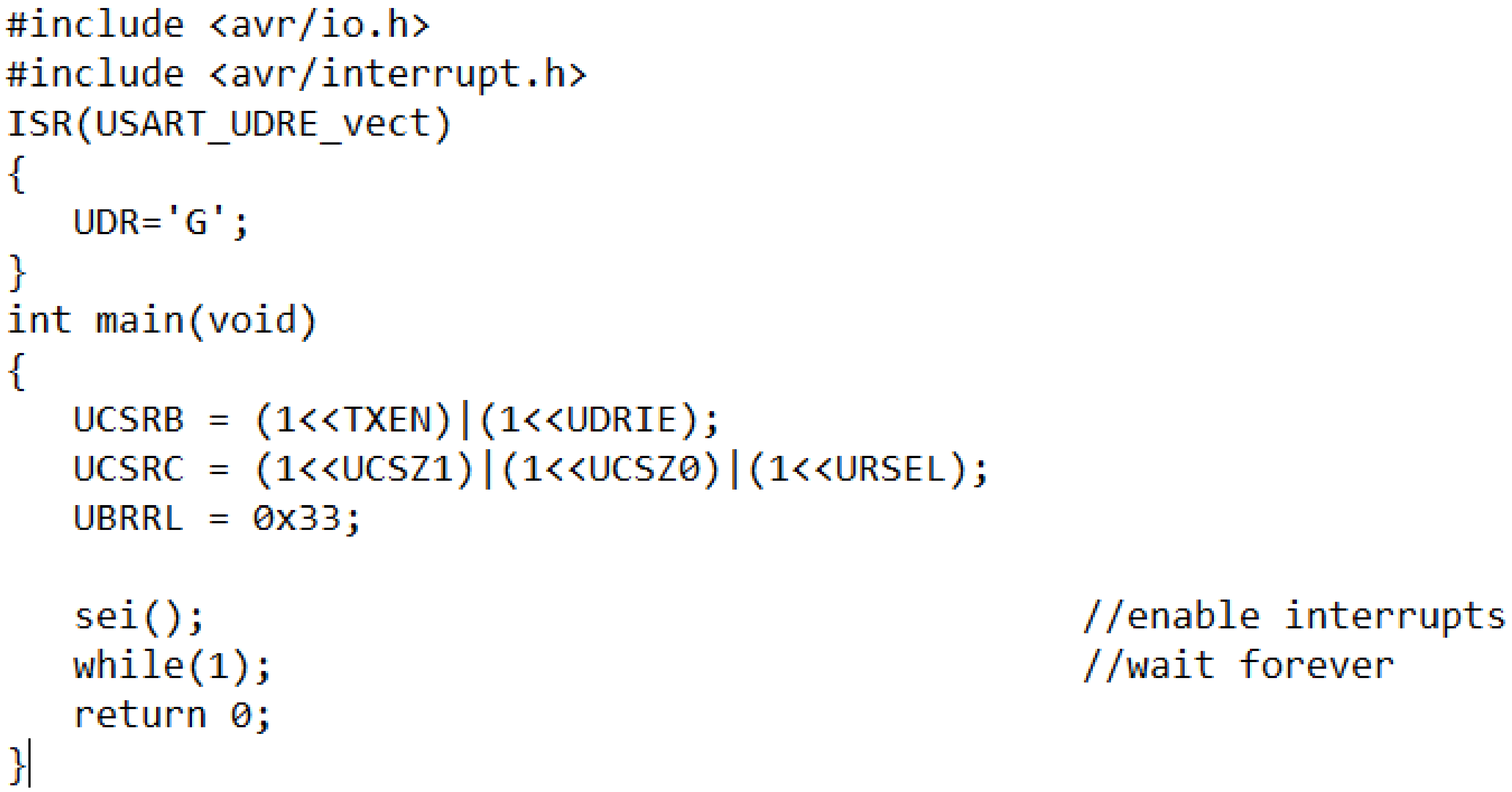
```
.CSEG                                       ;put in code segment
        rjmp MAIN                           ;jump main after reset
.ORG UDREaddr                               ;int. vactor of UDRE int.
        rjmp UDRE_INT_HANDLER               ;jump to UDRE_INT_HANDLER
.ORG 40                                     ;start main after interupt vector

MAIN:
   ldi r16, high(RAMEND)                    ;initialize high byte
   out sph, r16                             ;of stack pointer
   ldi r16, low(RAMEND)                     ;initialize low byte
   out spl, r16                             ;of stack pointer
   ldi r16, (1<<TXEN)|(1<<UDRIE)            ;enable transmitter
   out UCSRB, r16                           ;and UDRE interupt
   ldi r16, (1<<UCSZ1)|(1<<UCSZ0)           ;sync., 8-bit
   out UCSRC, r16                           ;data no parity, 1 stop bit
   ldi r16, 0x33                            ;9600 baud rate
   out UBRRL, r16
   SEI                                      ;enable interrupts

WAIT_HERE:
   rjmp WAIT_HERE                           ;stay here

UDRE_INT_HANDLER:
   ldi r26, 'G'                             ;send 'G'
   out UDR, r26                             ; to UDR
   RETI
```

Example: Interrupt-based Data Transmit

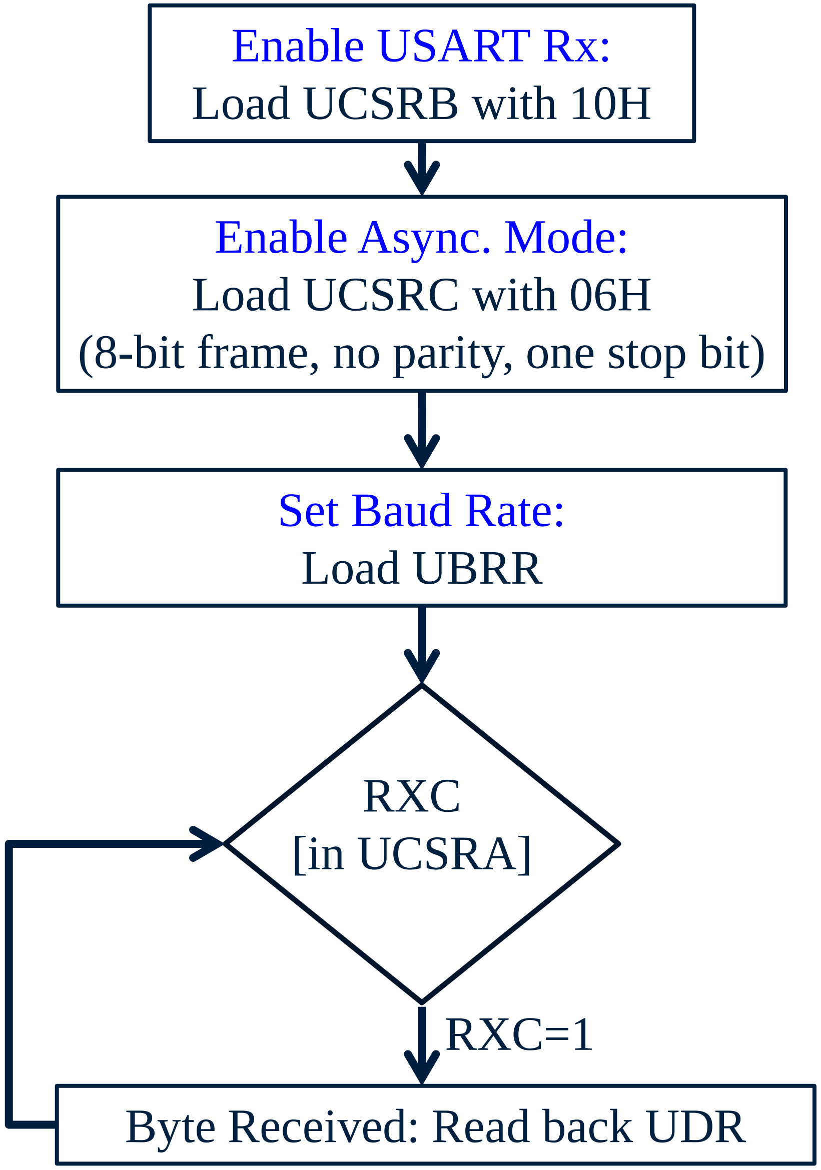


```c
#include <avr/io.h>
#include <avr/interrupt.h>
ISR(USART_UDRE_vect)
{
   UDR='G';
}
int main(void)
{
   UCSRB = (1<<TXEN)|(1<<UDRIE);
   UCSRC = (1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
   UBRRL = 0x33;

   sei();                                       //enable interrupts
   while(1);                                    //wait forever
   return 0;
}
```

Programming the AVR to Receive Data Serially

Programming the AVR to Receive Data Serially



Example: Receive Data Serially [Assembly]

```
    ldi r16, (1<<RXEN)                            ;enable receiver
    out UCSRB, r16
    ldi r16, (1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL)     ;8-bit data
    out UCSRC, r16                                ;no parity, 1 stop bit
    ldi r16, 0x33                                 ;9600 baud rate
    out UBRRL, r16
    ldi r16, 0xff                                 ;Port B is output
    out DDRB, r16
RCVE:
    sbis UCSRA, RXC                               ;is any byte in UDR
    rjmp RCVE                                     ;wait more
    in r17, UDR                                   ;send UDR to R17
    out PORTB, r17                                ;send R17 to Port B
    rjmp RCVE                                     ;do it again
```

Example: Receive Data Serially [C]

```
#include <avr/io.h>                                         //standard AVR header
int main(void)
{
        DDRA = 0xFF;                                        //Port A is output
        UCSRB = (1<<RXEN);                                  //initialize the USART
        UCSRC = (1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
        UBRRL = 0x33;
        while(1)
        {
                while(! (UCSRA & (1<<RXC)));                //wait until new data
                PORTA = UDR;
        }
        return 0;
}
```

Example: Receive Data Serially using Interrupt [C]

```
ISR(USART_RXC_vect)
{
        PORTB = UDR;
}

int main(void)
{
        DDRB = 0xFF;                                    //make PORT B an output port
    UCSRB = (1<<TXEN) | (1<<RXEN);                      //enable receive and RXC init
    UCSRC = (1<<UCSZ1) | (1<<UCSZ0) | (1<<URSEL);
    UBRRL = 0x33;
        sei();
//enable interrupts
    while (1);                                          //wait forever
        return 0;
}
```

Recommended Reading

- The AVR Microcontroller and Embedded Systems: Using Assembly and C by Mazidi et al., Prentice Hall
  - Chapter 11 -> 11.3 and 11.4

THANK YOU